Mutokamwoyo Cloud

Juan Jose Montiel Cano

Sergio Alfonso Semedi Barranco

Alba Maria Montero Monte

Adrian Martinez Jimenez

TRABAJO FIN DE GRADO EN ESPECIALIDAD EN INGENIERÍA INFORMÁTICA

12/05/2017

Director: José Luis Vázquez Poletti

Codirector: José Manuel Velasco Cabo

Autorización

Los abajo firmantes, Sergio Alfonso Semedi Barranco, Alba María Montero Monte, Adrián Martínez Jiménez y Juan Jose Montiel Cano, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la presente memoria: “Mutakamwoyo Cloud”, como los contenidos audiovisuales y la documentación desarrollados durante el curso académico 2016-2017 bajo la dirección de los profesores José Luis Vázquez-Poletti y Jose Manuel Velasco Cabo en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Alumnos

Sergio Alfonso Semedi Barranco	Alba María Montero Monte
Adrián Martinez Jimenez	Juan Jose Montiel Cano

Directores

Jose Luis Vazquez Poletti	José Manuel Velasco Cabo

Madrid, 16 de Junio de 2017

Índice

Agradecimientos

En primer lugar, agradecer como no a Jose Luis Vázquez Poletti y Jose Manuel Velasco Cabo por apoyarnos y guiarnos a lo largo del desarrollo de este proyecto, por haber llevado nuestra idea más allá, convirtiendo nuestro proyecto en una gran iniciativa para llevar recursos donde no los hay.

Agradecer también a la ONG *Mutoka Mwoyo* por haberse reunido con nosotros para facilitarnos información, sobre los medios y los recursos necesarios para poder convertir este proyecto en una realidad.

Finalmente, nuestro más sincero agradecimiento a la organización del Concurso de Software Libre de Sevilla, por haber apoyado nuestro proyecto y considerarnos ganadores del premio al mejor proyecto cloud.

A todos vosotros gracias, por haber creído en nosotros.

Resumen

Internet, una de las herramientas más populares y de mayor capacidad de comunicación e información que existen. La riqueza de Internet en contenidos es mucho mayor que la de cualquier biblioteca del mundo, además de estar actualizándose continuamente y lo mejor de todo al alcance de nuestra mano,de manera inmediata, lo único que necesitamos es un ordenador con conexión. Su contenido es tan diverso que podríamos afirmar que si buscamos algo en ella y no lo encontramos, es que no existe. Uno de los impactos más significativos de Internet es el que tiene sobre la educación, proporciona aprender con libertad, colaborar de forma interactiva con otros usuarios o estudiantes y compartir los recursos. Si a todo esto le añadimos la computación en la nube descubrimos un nuevo mundo de posibilidades. Las plataformas de educación como por ejemplo aulas virtuales para universidades a distancia han facilitado el acceso a la educación a nivel mundial.

Hoy en día tener conexión a Internet se considera igual de necesario que la electricidad o el consumo de agua en nuestras casas.Pero como siempre sucede lo que nosotros damos por sentado y consideramos esencial en nuestro día a día no es así para las personas que viven en países con recursos limitados o subdesarrollados, en los que ni siquiera las necesidades realmente básicas y esenciales como la comida o el agua pueden garantizarse. *Mutokamwoyo Cloud* pretende solventar esta falta de recursos educativos gracias a estas dos herramientas: Internet y la nube.

Palabras clave:

- Computación en la nube
- Educación
- SaaS (Software como servicio)
- Voluntariado
- Software Libre
- Proveedor de servicios
- Cliente
- Servidor
- ASP

Abstract

The Internet, one of the most popular and powerful communication tool and information provider of the world. Internet's content wealth is much bigger than any library in the world, besides it is continuously updating and best of all, it's on the reach of our hands, immediately, all you need is just a computer with a network connection. Its content is so diverse that we could say, "if we're looking for some kind of information and we are unable to find it, it will probably not exist".

Internet usage in education has seen tremendous growth, not only because of the freedom learning, but the faster interaction with another users and sharing resources. If we add Cloud computing to the mix it opens up a whole new world of possibilities. Learning platforms such as virtual classrooms for distance universities have provided an easier and worldwide way access to education.

Nowadays have an Internet connection is considered as needed as electricity or current water in our homes. However, as always happens, things we take for granted or required, (from our point of view) are not for people who live in underdeveloped and limited resources countries, in which even the truly and most basic essential needs like food or water can't be granted.

Mutokamwoyo Cloud tries to solve these lack of education resources thanks to these two tools: The Internet and Cloud Computing.

Keywords:

- Cloud Computing
- Education
- SaaS (Software as a Service)
- Charity
- Content Provider
- Free Sofware
- Client
- Server
- ASP

Capítulo 1: Introducción

.

Imagen 2

«Mutoka Mwoyo Cloud» es un proyecto de voluntariado coordinado por varios estudiantes de la Facultad de Informática de la Universidad Complutense de Madrid que cursan su último año académico de Ingeniería informática (2016-2017). Tiene como objetivo implementar una red Intranet en el pueblo congoleño de Ngandanjika para poder abastecer de información básica a su población y, en especial, a los médicos del hospital «Virgen de Guadalupe» y a los estudiantes y profesores del colegio «La Robertanna».

La red Intranet se provisionara gracias a un sistema de Cloud Computing, con Mcloud pretendemos diseñar, implementar y desplegar ese sistema.

1.1 ¿Quiénes somos?

Los estudiantes que trabajan de forma voluntaria en el proyecto «Mutoka Mwoyo Cloud son los siguientes:

- Sergio Alfonso Semedi Barranco (4º de Grado en Ingeniería Informática).
- Alba María Montero Monte (4º de Grado en Ingeniería Informática).
- Adrián Martínez Jiménez (4º de Grado en Ingeniería Informática).
- Juan José Montiel Cano (4º de Grado en Ingeniería Informática).

Contamos con la asistencia y asesoramiento de los siguientes profesores universitarios para ejecutar el proyecto:

- José Luis Vázquez Poletti
- José Manuel Velasco Cabo

Departamento de Arquitectura de Computadores y Automática en la Universidad Complutense de Madrid.

1.2 Motivaciones

Actualmente en el poblado *Ngandanjika* consiguen un acceso a Internet muy limitado y caro, accesible solo desde un edificio aislado, en el que debido al alto precio de la conexión, el acceso a Internet solo puede estar activo a altas horas de la madrugada. Esto hace que la conexión además de ser mala no sea muy útil debido a que solo funciona en las horas de sueño.

.

Imagen 3

Projet Ditunga es una Organización no gubernamental que intenta ayudar con pequeñas acciones que contribuyan a mejorar la delicada situación que vive el país. Ha sido la encargada de implantar y gestionar las instalaciones mencionadas anteriormente. Contamos con su colaboración para poner en marcha nuestra idea y conocer las condiciones del entorno donde implantar nuestro sistema.

MUTOKA MWOYO significa “hola blanco” y son las primeras palabras que aprende cada nuevo viajero al pisar por primera vez la tierra donde se habla *tshiluba*. Todos los niños lo gritan en cuanto nos ven pasar cerca de ellos, es su primera forma de dirigirse ante esos extranjeros que han venido a visitarles. Y eso es lo primero que nosotros queremos también hacer, saludar a todo el mundo occidental!

1.3 Objetivos

Con este proyecto se pretende implementar un servicio a disposición de todos que ayude a gran escala en ámbitos culturales, educativos médicos y de entretenimiento. La principal herramienta para este propósito será Internet, pretendemos entonces que gracias a nuestro proyecto todos los niños, profesores, médicos y cualquier ciudadano pueda tener acceso a cualquier hora del día.

El escenario principal es el cibercafé, en que instalaremos nuestro sistema para proporcionar servicio a los ciudadanos de la zona, con la posibilidad de extenderlo más adelante al hospital de la zona y a la escuela.

Ademas Mutokamwoyo Cloud sera un proyecto de codigo libre, con el objetivo de crear un sistema moderno Cloud que pueda ser reutilizable para casos de uso similiares al que se situa nuestro proyecto, hablaremos de esto en la seccion Estado del arte.

1.4 Medios utilizados

Debido a las circunstancias que nos encontramos en el poblado de *Ngandanjika* y más concretamente en la zona del cibercafé recurriremos a los siguientes medios:

- **Conexión satélite a Internet**: con un ancho de banda de 128 a 384 kbps, lo cual dificulta el acceso a Internet así como la obtención de grandes archivos.
- **Servidor en el Congo**: que gestione las peticiones de los usuarios de la red de una manera útil debido a las limitaciones que nos encontramos. Para ello recurrimos al diseño de uno propio buscando optimizar el consumo y la gestión de los recursos a nuestro alcance.

- **Servidor en España**: que se encargará de recibir y resolver las peticiones del servidor en el Congo con el fin de optimizar el uso de la red y de los recursos.

1.5 Funcionamiento

Explicado el problema y la situación, el proyecto es el siguiente, dado el gran problema de capacidad que tiene la red proporcionada por el satélite, el sistema se dedicará a guardar, durante el día, todas las peticiones que realicen los usuarios del sistema, de esta forma, todas las peticiones que no sean urgentes, podrán guardarse y no saturarán el sistema, dejando así todo el ancho de red a las peticiones urgentes que puedan surgir durante el día. Al llegar una hora determinada de la noche, cuando se calcule que el uso de la red está disponible, será cuando el sistema desencadenará todo su potencial, procesando todas las peticiones realizadas durante el día, para ello el sistema comprime lo máximo posible las peticiones y, a través del satélite, envía una petición con el paquete comprimido a Madrid, donde se encontrará el servidor encargado de procesar y gestionar estas peticiones para transformarlas en el contenido solicitado, posteriormente toda esa información se separará por contenido, asignando además un formato acorde con el mismo y se enviará con la máxima comprensión posible de vuelta al sistema instalado en el Congo. Cuando los paquetes comprimidos lleguen al Congo, el sistema los separará por contenido y se procederá a la descompresión del mismo, de esta forma, todas las peticiones solicitadas por los usuarios durante el día, se convertirán en información la cual se presentará en un sistema que actúa como repositorio en el cual se podrá gestionar, consultar, y acceder al contenido que estará debidamente separado por categorías dependiendo del mismo.

Chapter 1: Introduction

.

Image 2

«Mutoka Mwoyo Cloud» is a volunteer project coordinated by many students from Universidad Complutense de Madrid (UCM), as part of their final project in the last year of Computer Science Engineering (2016-2017). The main goal of this project is to implement an intranet network in Ngandanjika a little village of Congo (Africa) to be able of providing basic information to the population, especially to doctors of «Virgen de Guadalupe» hospital and teachers and students of «La Robertanna» school.

The intranet network will be provided thanks to a Cloud Computing system, with MCloud we try to design, implement and deploy this system.

1.1 Who are we?

The students who work in the «Mutoka Mwoyo Cloud» project in a voluntary way are:

- Sergio Alfonso Semedi Barranco (4º course of Computer Science Engineering).
- Alba María Montero Monte 4º course of Computer Science Engineering).
- Adrián Martínez Jiménez (4º course of Computer Science Engineering).
- Juan José Montiel Cano (4º course of Computer Science Engineering).

We have the assistance and advice of two college professor to carry out the project:

- José Luis Vázquez Poletti
- José Manuel Velasco Cabo

Distributed Systems Architecture Group at Universidad Complutense de Madrid.

1.2 Motivations

Currently, in the village of *Ngandanjika* they only can get a very limited and expensive access to the Internet, in a remote building, besides due to the high price of connection, Internet access can only be able at late-night hours. This fact makes the connection not only poor but useless because only works effectively during periods of sleep.

.

Image 3

Projet Ditunga is a non-governmental organization (NGO) that tries to help with little actions that contribute to improving the fragile situation in the country. This NGO has been responsible fro building the facilities we mention before. Thanks to them we have been able to carry out our idea and start this project, they have provided us very useful information about the environment where we want to implant our system.

MUTOKA MWOYO means "hello white" these are the first words that every traveler who first arrive in this land learn, this language is known as *tshiluba*. Every child in there shout these words when they see the travelers passing net to them, it is the way to say hello and welcome to those who have come to visit them. And that is what we want to do too, just say hello to the occidental world!.

1.3 Goals

With this project, we try to implement a service accessible to everyone to help in a cultural, educational, medical way and of course provide entertainment too. The main tool for this purpose will be The Internet, we want our project to provide every child, teacher, doctor or ordinary people access to it at every time of the day.

The main target is the cyber cafe in which we will install our system to provide the service to the population of the village, with a chance to spread it to the hospital and the school once this main service works properly.

In addition, Mutokamwoyo Cloud will be a free code project, in order to create a modern Cloud system that can be re-used for similar situations.

1.4 Used Media

Due to the environment conditions, we found at the village of *Ngandanjika*, at the cyber cafe to be more specific we will use the next resources:

- **Internet connection via satellite**: 128 a 384 kbps, which makes Internet access and big files downloading very tedious and almost impossible task.
- **Congo server**: it will manage users petitions in an effective way due to the limitations we have. In order to do that we have designed a server of our own trying to optimize the consumption and the management of the resources we have.

- **Spain Server**: it will be responsible for receiving and resolving the incoming petitions from the Congo Server, in that way we will optimize the network use and resources.

1.5 System operation

Once we have explained the problem and the situation, the project explanation is this, due to the bid deal of the system capability and the network provided by the satellite, the system will be storing, during the day, each petition made by our system users, in that way, all the non-urgent petitions, could be stored and the system won't be overflowed, this will let the network available for urgent petitions. At a determined time of the night when the use of the network and the connection is the fastest, our system will make it best, processing all the petitions that have been made during the day, compressing as much as possible that petitions and throw the satellite, a compress package petition will be sent to the server located in Madrid. When the package arrives the server that manages the petitions will decompress the package and transforms the petitions into the content requested, after that, all that information will be divided into categories depending on the type of content, and compress it once again to send it back to the server located on Congo. Once the package arrives there, the system will decompress it, divide the content and all those petitions that the users made during the day will become into information. This information will be presented in a system that acts as a repository in which users will be able to manage, read, and access to it depending on the category and the type of content.

Capítulo 2: Estado del arte

2.1 Concepto de *Cloud Computing*

A pesar de ser una concepto que cada vez se está haciendo más popular la idea del *Cloud Computing* sigue siendo, para algunos, una "tecnología" difícil de comprender. Para dejar claro este concepto empecemos con la definición, *Cloud Computing no es una tecnología*.Es nuevo paradigma informático que permite ofrecer distintos servicios a través de la red.

<> (No hay nube, sólo ordenadores de otras personas), es una frase que explica muy bien el funcionamiento del *Cloud Computing*. La información se almacena de forma permanente en servidores de Internet repartidos por todo el mundo y son los que se encargan de atender las peticiones en cualquier momento.

Dentro del concepto *Cloud*, podemos distinguir tres tipos según el tipo de servicio que ofrecen:

- ***SaaS*** (Software as a Service). Por ejemplo *Google Drive*.
- ***IaaS*** (Infrastructure as a Service). Por ejemplo *Microsoft Azure*.
- ***PaaS*** (Platform as a Service). Por ejemplo *ec2*.

2.2 internet.org o Free Basics

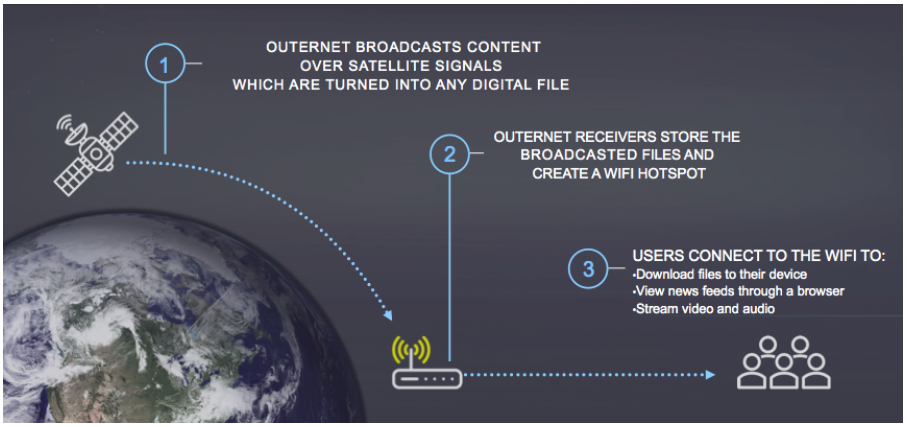
Este proyecto fue lanzado por Facebook el 20 de agosto 2013, su principal finalidad es llevar ciertos servicios de internet a países en vías de desarrollo, estos países no tiene la posibilidad de acceder a servicios básicos de internet. Entrando mas en detalle de lo que pretende hacer este proyecto, es llevar una aplicación móvil que permita acceder a ciertos servicios a estos países, la forma de transportarla no es mas que por redes móviles, para ello el proyecto se unió con ciertos proveedores de internet móvil para poder hacer llegar estos servicios a los países anteriormente nombrados. Para poder realizar este servicio se decidió lanzar un satélite que proporcionase esa conexión, SpaceX fue la empresa encargada de realizar esta tarea, por desgracia, este satélite nunca llego a estar en órbita ya que tuvo un accidente.

Imagen 4

Ahora bien, este sistema no ha sido muy bien aceptado en estos países, ya que aunque proporciona acceso a ciertas aplicaciones de forma gratuita, no es gratuito de forma completa, ya que Facebook puede acceder a todos los datos que pasen por su sistema y vender esta información. Este es el motivo principal de su rechazo en varios países del mundo, por ello algunos países como Egipto o India ya han bloqueado el uso de este sistema en sus países.

2.3 Outernet

La empresa neoyorquina MDIF (Media Development Investment Fund) lanzó el proyecto Outernet Inc. cuya idea era permitir el acceso gratuito a Internet vía satélite. Colocaron varios microsatélites CubeSats que orbitan la Tierra para dar acceso público al ciberespacio,(de ahí el nombre de outernet). Su idea era permitir que internet llegara a paises con recursos limitados o en los que la censuda impide el acceso libre a Internet, como Cora del Norte o China, mediante una constelación de satélites de bajo coste.



Sudebut fue en 2015 y aunque ofrecer la web vía satélite a lugares a los que no llegan las líneas de comunicación habituales no es una idea nueva, por el momento es una solución cara y que en muchos casos tampoco asegura una gran velocidad. Mantener outernet en funcionamiento tendría unos costes que habría que seguir sufragando año a año.

https://www.mdif.org/outernet-providing-information-to-the-world-from-outer-space

Imagen 5

2.4 Project Loon (by Google)

Project Loon es un proyecto que se conoce hace tiempo que fue ideado por Google a través del cual, usando globos, pretenden darle Internet gratuita a todo el mundo, centrandose sobre todo en zonas remotas o rurales donde no hay cobertura y, también, para que las personas puedan volver a tener Internet después de una catástrofe.

Los globos, pueden dar conectividad a un área terrestre de aproximadamente 40km de diámetro, flotan en la estratósfera, por encima del nivel que utiliza el tráfico aéreo comercial y por encima de donde se producen los fenómenos meteorológicos. Entre 18 y 27km de altura. Desde Google se asociaron con empresas de telecomunicaciones para compartir el espectro celular y así permitir que las personas se conecten a la red del globo directamente desde sus teléfonos y cualquier dispositivo que tenga conectividad móvil 4G/LTE.

Imagen 6

La energía usada por los dispositivos de los globos es impulsada a través de sus dos paneles solares que están en sus costados, posicionados en un ángulo empinado para que puedan capturar de forma efectiva la luz solar, en horas de sol pueden alcanzar los 100W de eergía permitiendo que sigan siendo funcionales durante la noche.

Google comenzó el 17 de junio de 2013 un programa piloto en Nueva Zelanda, donde lanzaron a la estratosfera 12 globos.

Capítulo 3: Descripción

Con Mutokamwoyo Cloud pretendemos llevar a cabo un proyecto basado en el nuevo y famoso paradigma de Cloud Computing que sea capaz de hacer llegar recursos necesarios en sitios donde seria imposible conseguirlos de otra manera.

Para poder llevar a cabo este proyecto hemos tenido que separar el proyecto en varias zonas, separadas geográficamente con un rol que interpretar en nuestro sistema Cloud y diseñar un protocolo de comunicaciones para ese sistema que nos sea válido.

Es importante añadir, que además de separar las distintas zonas del sistema, Mutokamwoyo Cloud también planifica el diseño hardware y la arquitectura básica de la red donde se implementará, por lo que el trabajo del proyecto a su vez tiene dos partes:

- **Software:** El sistema operativo y el código de la aplicación que realizará la interacción con las distintas zonas y la definición del protocolo de comunicación.
- **Hardware:** Máquinas, servidores que harán de host del sistema operativo junto con el código ya mencionado, además de los dispositivos necesarios para que sea posible la comunicación entre las zonas.

Para empezar a describir nuestro proyecto hemos decidido separar la descripción en tres secciones pricipales:

- **Software**: Donde se describen todas las tecnologías utilizadas para la realizacion de este proyecto.
- **Hardware**: Donde se describen cuales son los dispositivos hardware que se han decidido utilizar para la implantación del proyecto.
- **Sistema**: Donde se describe en detalle los aspectos tecnicos del proyecto.

3.1 Software

Al consistir en un proyecto cloud podría decirse que la parte software y la parte hardware son las dos de igual importancia. Sin embargo, como en este proyecto de fin de grado estamos **implementando nosotros mismos** el software que vamos a utilizar y escribiéndolo de cero, supone una carga de trabajo mucho mayor.

Para ello, un controlador de versiones es indispensable, nosotros hemos utilizado **GIT** hemos alojado todo el código en https://github.com/MutakamwoyoCloud y hemos aprovechado las diferentes herramientas que proporciona GitHub y el poderoso controlador de versiones GIT para organizar todo el desarrollo.

Una vez elegido el hosting, y el sistema controlador de versiones nos toca pensar en Frameworks y lenguajes, en MCloud nos hemos decidido por usar Python y JavaScript como lenguajes principales:

Javascript

Imagen 7

Por lo popular que es y la facilidad que tiene para el desarrollo open source, nos permite tener una fuente de documentación muy amplia ademas de una variedad de frameworks muy buena.

Con él hemos decidido usar famosos frameworks como pueden ser **Node** o **React**, hablaremos sobre ellos desde un punto de vista más técnico en las secciones correspondientes de la *zona cliente*.

ReactJS.

.

Imagen 8

El frontend está desarrollado con *ReactJS* una librería *JavaScript* para el diseño de interfaces de usuario con un diseño modular basado en componentes. En lugar de escribir densos archivos de código podemos escribir varios trozos más pequeños y reutilizables, esto nos proporciona una forma sencilla y rápida de desarrollar aplicaciones de una sola página (SPA).

Características principales de ReactJS.

JSX es una extensión de *JavaScript* que fue escrita para ser utilizada con React. Su sintaxis muy parecida a HTML. Cuando decimos que *JSX* es una extensión de *JavaScript* quiere decir que no es un *JavaScript* válido, los navegadores no pueden leerlo. Si un fichero *JavaScript* contiene *JSX* entonces este debe compilarse. Esto significa que antes de llegar al navegador un compilador traduce el JSX a JavaScript.

SPA(aplicaciones de una sola página). Como su propio nombre indica es una aplicación en la que todo el contenido y las vistas son cargadas en la misma página. Esto rompe con la arquitectura y la forma de construir aplicaciones web a la que estábamos acostumbrados en las que por cada cambio que se hacía el navegador mostraba un página diferente. Las *SPA* no sólo permiten mejorar la experiencia de usuario si no que permiten un mantenimiento más sencillo y una reducción significativa en cuanto al almacenamiento de datos.

VIRTUAL DOM La manipulación del *DOM* (Document Object Model) es la arteria principal de las web interactivas. Desgraciadamente es también mucho más lento que la mayoría de las operaciones *JavaScript* ya que la mayoría de *frameworks* lo actualizan más de lo necesario. Aquí es donde React hace su magia. Podríamos considerar el *DOM VIRTUAL* de React como una copia ligera del *DOM*. Cuando un elemento se renderiza el VIRTUAL DOM se actualiza completamente y React lo compara con el DOM real actualizando únicamente los elementos del *DOM* que han cambiado.

ROUTING Cuando desarrollamos una *SPA* la parte difícil llega cuando hablamos del enrutado ya que sólo existe una página y lo único que cambia es el contenido, debemos asegurarnos de que la url que mostramos en barra de direcciones se corresponde con el contenido y el botón de atrás de la navegación funciona correctamente. Necesitamos mapear las url a páginas que no son físicas, si no a componentes de vista. Para ello hacemos uso de *ReactRouer* librería de *JavaScript*.

Nodejs

Imagen 9

Una vez explicada la interfaz de nuestra aplicación falta el núcleo de esta. Al núcleo que procesa todo y es el corazón de la zona cliente lo llamamos Core.

El core de MCloud como ya sabemos también esta construido en *JavaScript* y debido al alto populismo y la cantidad de artículos que existen sobre el, el framework utilizado es *NodeJS* que nos deja unas nuevas librerías pontentes y fiables para trabajar con *JavaScript* del lado del servidor.

Que es?

Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

Python

Imagen 10

Elegimos Python por ser un lenguaje simple, muy fácil de leer, algo muy relevante en un proyecto de software libre. Además al ser un lenguaje tan popular al igual que *JavaScript* nos da la posibilidad de tener una fuente de documentación muy amplia.

Con él hemos usado librerías como Watchdog, o APIs para poder sacar información mas fácilmente de Wikipedia, hablaremos de ellas en sus correspondientes capítulos.

Software Libre y gestión de librerías externas

En Mutokamwoyo Cloud creemos en el modelo de software libre, y sobre todo para un tipo de proyecto donde la caridad y ayudar a los demás es lo mas importante. Este paradigma comunitario encaja totalmente con nuestro proyecto y nos ayuda a la hora de trabajar, ya que podemos aprovechar todas las características de github y dejamos el código libre para que cualquier persona pueda colaborar en el proyecto de la forma que quiera.

Otra de las características mas importantes del software libre que hemos podido aprovechar es la compartición de código, gracias a gente que tiene la misma visión que nosotros a la hora de construir software, hemos podido usar alguna librería externa junto con dos gestores de paquetes principales:

NPM

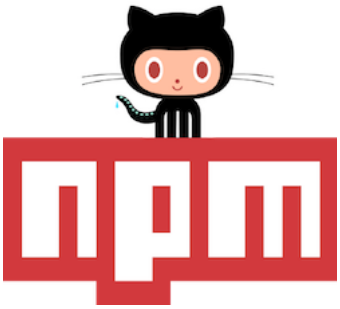


Imagen 11

Node package manager, es el gestor de paquetes por defecto de nodeJS, y seguramente el gestor mas utilizado por la facilidad que tienes para implementar cualquier proyecto sobre el tuyo.

Los paquetes principales que hemos usado con este gestor:

```
    "archiver": "^1.2.0",
    "babel-core": "^6.18.2",
    "body-parser": "^1.16.0",
    "concurrently": "^3.1.0",
    "decompress": "^4.0.0",
    "express": "^4.14.0",
    "fixed-data-table": "^0.6.3",
    "foundation-sites": "^6.3.0-rc1",
    "fs": "0.0.1-security",
    "ftp": "^0.3.10",
    "jquery": "^3.1.1",
    "mongodb": "^2.2.24",
```

```
    "node-schedule": "^1.2.0",
    "node-uuid": "^1.4.7",
    "react": "^15.4.0",
```

PIP

Pip es el gestor de paquetes de python, ofrece una facilidad parecida de uso y es de gran ayuda a la hora de instalar módulos externos y no tener problemas con dependencias.

Los paquetes que hemos usado con pip son:

```
watchdog==0.8.3
wikipedia==1.4.0
pytube==6.2.2
scrapy==1.4.0
```

3.2 Hardware

Debido a las circunstancias que nos encontramos en el poblado de Ngandanjika, más concretamente en la zona del cibercafé, zona designada para las labores relacionadas con internet, recurrimos a utilizar los siguientes medios:

- Conexión vía satélite a internet, con un ancho de banda de 128 a 384 kbps, lo cual dificulta el acceso a internet así como la obtención de grandes archivos.
- Un servidor que gestione las peticiones de la red de una manera útil con los medios que disponemos. Para ello recurriremos al diseño de uno propio buscando optimizar el consumo y la gestión de los recursos a nuestro alcance.
- Antenas WiFi de baja frecuencia para conectar redes a grandes distancias, ya que debemos plantear una etapa del proyecto en la que conectaremos el cibercafé al hospital. Un servidor alojado en España que se encargaría de realizar las peticiones del servidor en el Congo con el fin de optimizar el uso de la red y de los recursos.

3.2.1 Material y consideraciones

Debido a las dificultades a las que se tiene que enfrentar el proyecto, como la escasez de electricidad y el bajo presupuesto, optamos por la construcción de la intranet mediante miniordenadores de bajo coste y repetidores tipo Airfiber Wireless.

Para elegir el modelo de miniordenador, buscamos la opción más ajustada en relación potencia-precio del mercado (vease [Figura 1]). Elegimos la Odroid-XU4 ya que satisface las necesidades de potencia del proyecto teniendo un precio ajustado. Las bondades de la Odroid-XU4 han sido demostradas en numerosos benchmarks, donde consigue resultados superiores a la Rapsberry Pi 2 y la Odroid-U3.

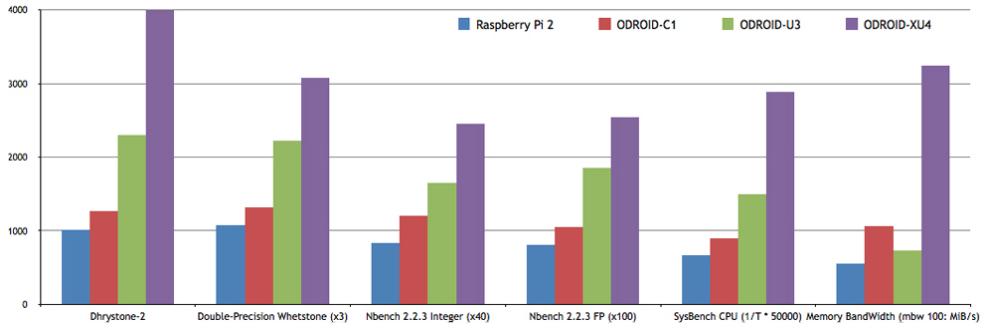


Figura 1: Benchmarks entre miniordenadores

Especificaciones Odroid-XU4:

```
Procesador: Samsung Exynos5422 Cortex™-A15 2Ghz and Cortex™-A7 Octa core CPUs
GPU: Mali-T628 MP6
Memoría RAM: 2Gbyte LPDDR3 RAM (750Mhz, 12GB/s de ancho de banda, 2x32 bit bus)
Periféricos:
 2 x USB 3.0, 1 x USB 2.0
 Gigabit Ethernet port
 HDMI 1.4a for display
Toma de corriente: 5V - 4A
Sistema operativo: Ubuntu 15.4
Dimensiones: 82 x 58 x 22 mm approx. (peso: 60 g incluyendo ventilador / 38 g sin ventilador)
```

Especificaciones AirFiber Wireless Adapter:

```
Tasas de transferencía mayores a 1.4 Gbps
Álcance mayor de 13 Km.
```

A la hora de construir cada uno de los servidores utilizaremos 4 Odroid-XU4 por cada uno, de tal manera que cada Odroid se encargue de diferentes módulos del sistema. Con esto tratamos de aliviar la carga de trabajo que tiene que soportar el servidor, juntando los servicios menos demandados y dando prioridad a los servicios que más se usan.

Distinguimos dos etapas para la implementación *Hardware*: 1. **Implantación en el cibercafé**: Modelo básico de componentes mostrados en la [Figura 2] con un sólo puesto montado en el cibercafé con todos los servicios disponibles, sin utilizar repetidores ni estar conectado a otro lugar.

DESCRIPCIÓN	PRECIO/UD	CANTIDAD	TOTAL
 ODROID-XU4 (5V/4A EU PLUG)	74€	4	296€
 USB 3.0 TO SATA	15€	4	60€
 DISCO DURO 4TB	250€	4	1000€

TOTAL: 1371€

Figura 2: Presupuesto cibercafé

2. **Extensión del servicio**: Ampliación de la intranet a más lugares, lo que conllevaría el montaje de más mini ordenadores y de repetidores, nombrados en la [Figura 3], en cada uno de los puestos de la intranet. A continuación presentamos una lista con los componentes *Hardware* necesarios para llevar a cabo este despliegue:

DESCRIPCIÓN	PRECIO/UD	CANTIDAD	TOTAL
AirFiber Wireless Technology 	1500€	2	3000€
NUEVAS LOCALIZACIONES: Componentes fase 1: - ODROID-XU4 - 3.0 TO SATA - DISCO DURO 4TB Componentes fase 2 AirFiber Wireless Technology	1371€ + 1500€	----	2871€

TOTAL: 5871€

Figura 3: Presupuesto extension

3.2.2 Intranet

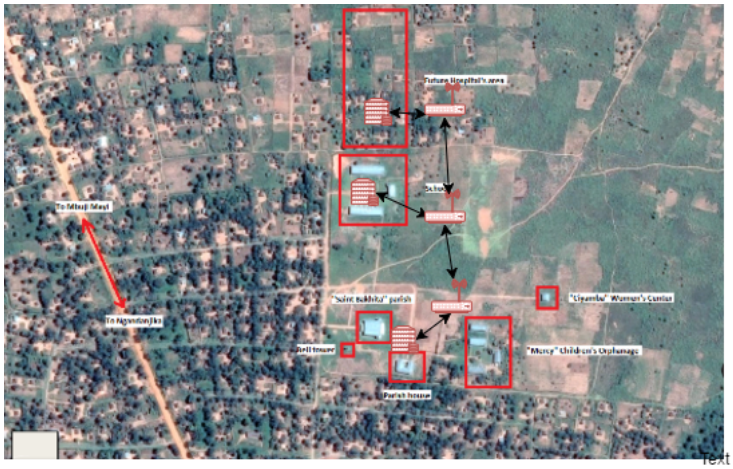


Figura 4: Mapa poblado(República democrática del Congo)

La intranet inicial estará situada en el cibercafé como podemos ver en el mapa de la [Figura 4], desde donde se podrá acceder a todos los servicios y se dispone de la conexión satélite.

En la fase de ampliación se pretende dar mayor cobertura a la zona, para ello se utilizarán repetidores Airfiber colocados según las características del terreno. Mediante estos repetidores se formará una intranet cuyo punto de conexión a internet se encuentre en el cibercafé. Las localizaciones a las que se pretende dar acceso en esta fase son: * Cibercafé * Colegio * Hospital

3.3 Sistema

En esta sección de la memoria se procederá a explicar en detalle todo lo relacionado con el sistema que se ha desarrollado, para empezar, este sistema esta separado en dos secciones principales como podemos ver en la [figura 5]:

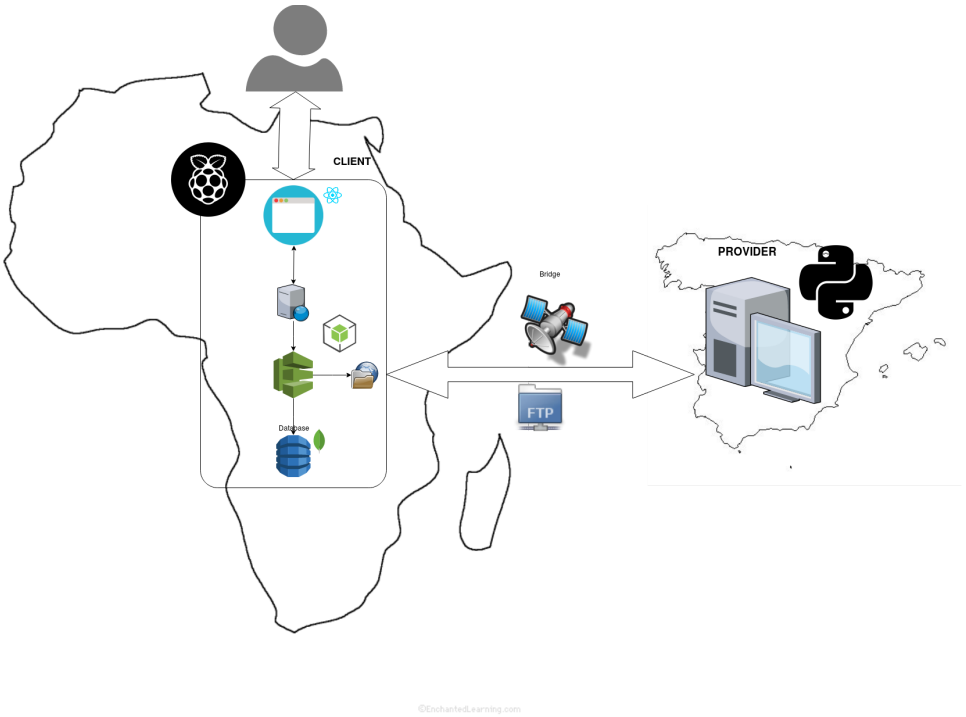


Figura 5: Esquema del sistema

- Zona cliente

En nuestro sistema, esta será la zona que no disponga de banda ancha y su población no tenga acceso a internet, en nuestro caso esa zona es el Congo, en ella dispondremos como mínimo de una máquina con acceso a datos a través del satélite de una forma muy lenta con alto coste y a unas horas determinadas del día.

El sistema operativo utilizado sera una distribución GNU/Linux como Ubuntu, la máquina correrá dos procesos para nuestra zona cliente, uno de interfaz gráfica que permite a otros clientes conectarse remotamente a la GUI de la aplicación y el proceso core que ejecutará las distintas acciones necesarias del sistema que corren por debajo (demonio).

Para desarrollar el software de la zona cliente como ya hemos comentado en el cápitulo anterior *JavaScript* será nuestro lenguaje principal, la zona cliente ademas deberá de tener algún tipo de conexión exterior por mala que sea, en nuestro caso usamos un satélite que ya se encuentra disponible y dispositivos hardware inalámbricos que nos permitan extender una intranet de la capacidad deseada.

- Zona Proovedor

Esta parte del sistema se instalará en una zona que tenga disponible banda ancha a internet como puede ser ADSL, VDSL o fibra óptica, en nuestro caso la zona proovedor se encontrará en una empresa de Madrid con disponibilidad de internet, el despliegue en fase de producción es recomendable que se haga en una maquina virtual o container para conseguir una capa de abstracción portable de esta parte de la aplicación.

El sistema operativo utilizado será una distribución de GNU/Linux como puede ser Debian o Ubuntu, sobre ella correremos un proceso de Python que se encargará de manejar el acceso real a los recursos solicitados por la zona cliente, aunque la zona cliente tenga una mala conexion y solo pueda acceder a internet a determinadas horas del día, la zona proovedor sin embargo estará activa las 24h y no tendrá ninguna restricción a la hora de acceder al contenido.

Funcionamiento

MCloud no solo funciona como un proveedor de internet en sitios donde hay difícil acceso, también gestiona el modo en el que se va a realizar la sincronización (vease [Figura 6]).

Como ya sabemos, el sistema está compuesto por varios módulos, un ejemplo podría ser Wikipedia y YouTube. Las peticiones de Wikipedia entonces al ser de finalidad didáctica deberían tener preferencia sobre las otras, al igual que el módulo Vademécum (medicina) que independientemente de las demás peticiones se actualizara automáticamente para que los usuarios puedan conseguir información sobre los medicamentos.

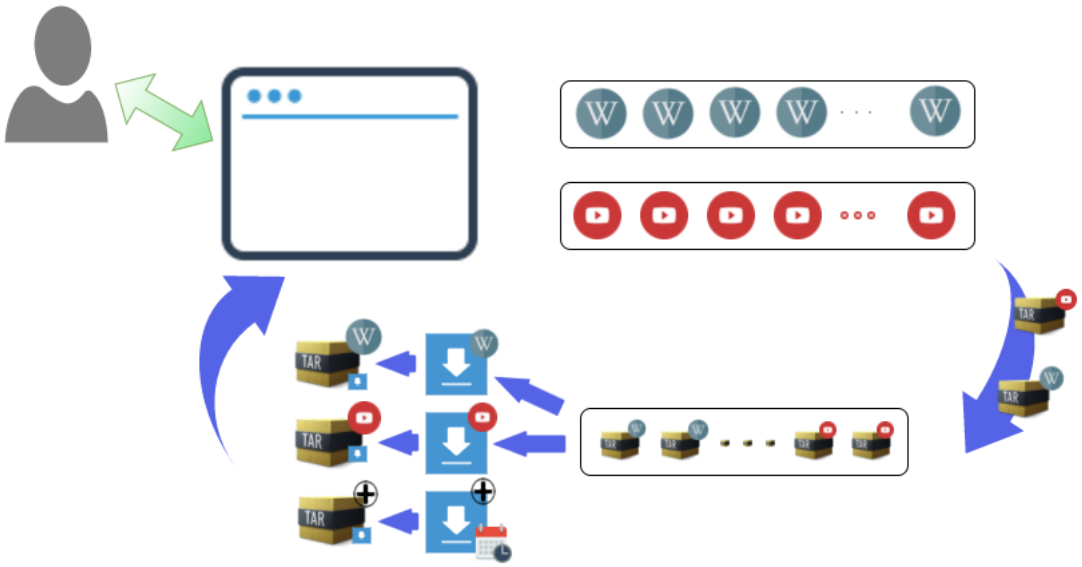


Figura 6: Esquema de funcionamiento del sistema

Debido al problema horario y que la descarga solo se efectuara a determinadas horas, MCloud tiene que contar con un sistema de cola que guarde preferencia sobre ciertos tipos de peticiones y como no sobre la hora a la que se pidió o lo que es lo mismo, el orden de prioridad sin contar el tipo lo da la hora en el que la realices. Esto no es suficiente si queremos tener en cuenta que habrá peticiones para el ocio, para la educación y para la medicina.

Para solucionar este problema MCloud tiene un sistema de prioridades. Todas las peticiones se encolaran de igual forma solo que las de medicina (Vademécum) se descargaran automáticamente antes de cualquier otro procesado y las de Wikipedia tendrán prioridad sobre las de YouTube.

Directorio

Organizar el directorio de trabajo de este sistema no es tarea fácil como podemos ver en la [figura 7], usamos varios lenguajes y en cada uno tiene que quedar bien separada cada parte correspondiente. Para poder agilizar esta tarea hemos usado la aplicación *Create-React-App* que nos brinda una arquitectura base para la parte de la *GUI*, está organizado con *Webpack* y nos permite tener separadas algunas zonas importantes como la carpeta public.

Una vez teniendo la base, nosotros hemos adaptado nuestra arquitectura creando las carpetas para los demás sistemas que queremos conectar como pueden ser el *back-end* o la parte proveedor, todo organizado de una forma en la que se intenta que nunca tengas problema en reconocer la zona en la que estas y su código.

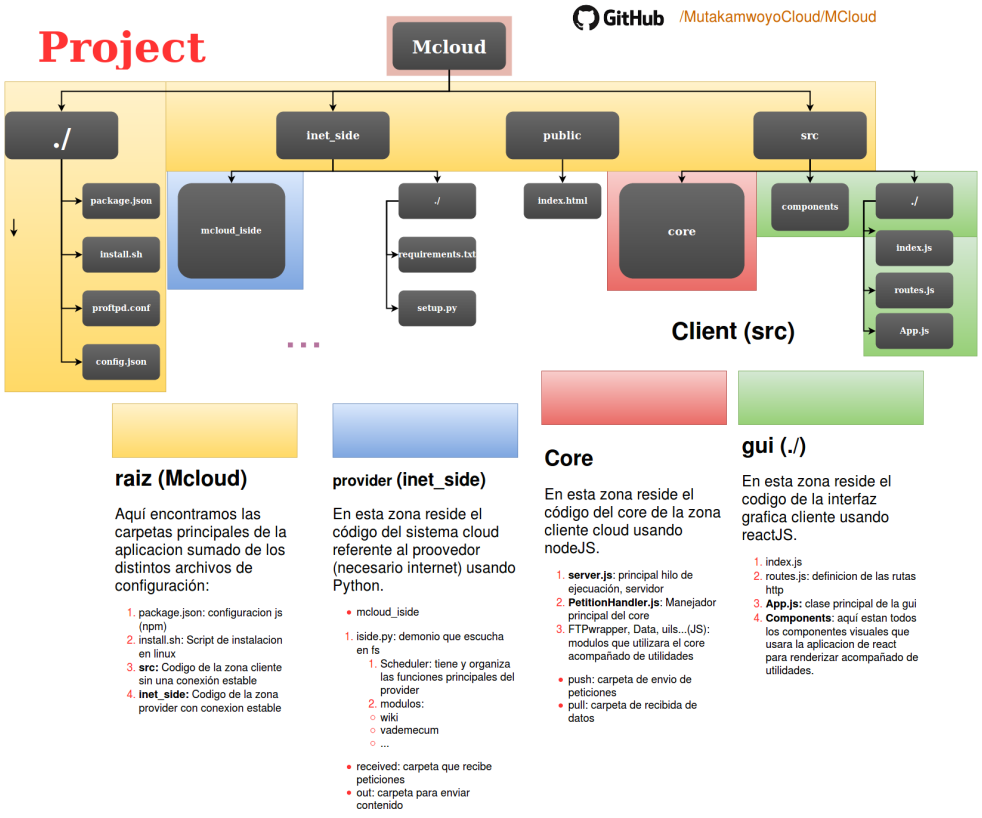


Figura 7: Esquema del directorio

3.3.1 Comunicación

Abstracción de FTP

Imagen 12

Ninguna de las dos zonas serviría para nada si no diseñamos un protocolo de comunicación para poder mantener el flujo de funcionamiento de MCloud.

En MCloud hemos optado por elegir FTP como la tecnología que se encuentra debajo nuestra para que se encargue de transferir el mínimo número de bytes de extremo a extremo.

Cita de Wikipedia:

FTP (siglas en inglés de File Transfer Protocol, 'Protocolo de Transferencia de Archivos') en informática, es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol), basado en la arquitectura cliente-servidor. Desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo.

Encapsulando la información

Imagen 13

Desde que hemos hablado de hacer toda la parte cliente en el lenguaje *JavaScript* y con *Mongodb* es evidente que un objeto de encapsulación como puede ser *JSON* era la respuesta evidente en cuanto a cualquier tipo de planteamiento que haga referencia a la forma de el envío de información.

En MCloud transmitiremos todo a base de *JSON* con la mínima información necesaria que sea posible. Finalmente para poder enviar la información de la manera mas óptima posible se decidió empaquetar todo en el formato tar.gz, esta decisión de diseño es debida a la escasa subida y bajada del que disponen las zonas clientes de MCloud.

Reglas

El conjunto de reglas que define nuestro protocolo es el siguiente:

Push

Operación de escritura sobre el proveedor, itera en orden sobre todas las peticiones que se encuentran disponibles y las envía al Proveedor para que este las pueda almacenar y posteriormente procesar.

Fetch

Operación que pregunta al proveedor por paquetes listos para descargar, nos asegura mantener el orden de peticiones según su relevancia a la hora de descargar paquetes.

Pull

Operación de bajada de datos, nos permite descargar del proveedor los distintos paquetes con toda la información necesaria, es necesario realizar antes la operación fetch para saber qué paquetes y en qué orden hay que descargar.

Flush

Final de la transacción MCloud de extremo a extremo, una vez que se ha hecho correctamente el intercambio de paquetes esta acción realiza un borrado de todos las peticiones o paquetes que sobran del protocolo de comunicación.

Zona cliente

Por si no ha quedado claro, a la hora de dividir nuestro sistema cloud, la **zona cliente** es la parte de nuestro sistema que tiene que ser instalada en una zona sin acceso a internet, ya sea por problemas geográficos, económicos o de no disponer de la tecnología suficiente. La zona cliente funcionará a su vez a modo de **Intranet** proporcionando así, además del servicio primario de la aplicación, otro servicio cloud **privado** que actuará sobre esa intranet de modo que cualquiera pueda tener acceso a esos recursos.

Organizar el Stack de la aplicación cliente utilizando frameworks como *React* o *Node* no es tarea facil, una de las formas que hemos utilizado para organizar el proyecto es usar la conocida librería *create-react-app*, con ella podemos organizar la estructura de *React* de una forma muy simple, además de que nos proporciona los scripts preparados, tanto como para desarrollar la aplicación, para testearla, pasarla a modo producción e incluso un servidor montado sobre el que organizar todo el *front-end* con *Webpack*.

Esto nos facilita mucho las cosas, sin embargo existe un **problema**. * Nuestra aplicación además de la parte visual de **React**, requiere un gran modelo por detrás (*back-end*) creado con ***Node***, pero nuestra aplicación ya tiene *Webpack* por defecto y no habría forma de comunicar las dos partes.

Para resolver esto, como podemos ver en la [figura 8] finalmente se ha optado por una arquitectura en la que el usuario al interactuar con el *front-end* de la aplicación manda proxy requests al sistema *back-end* funcionando en otro proceso del servidor.

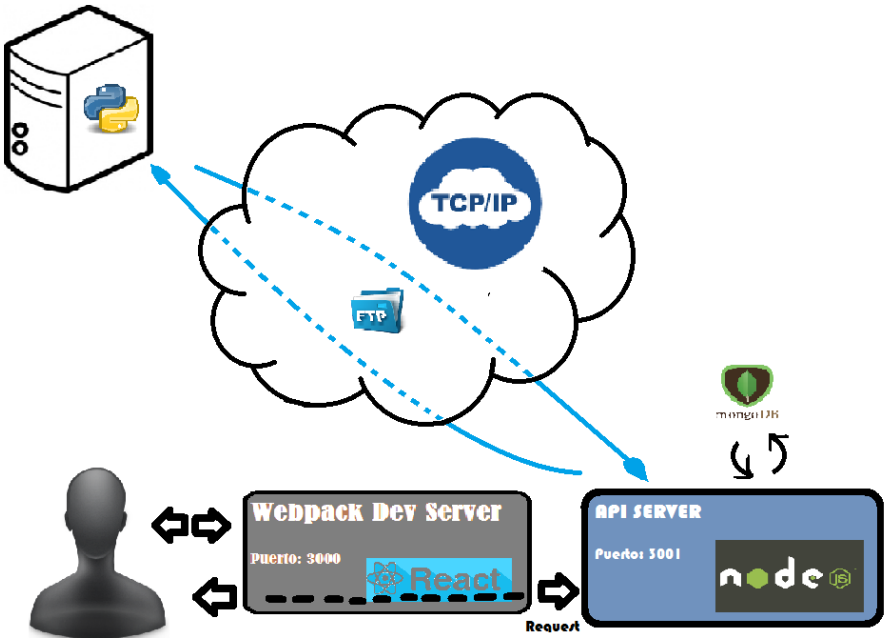


Figura 8: Peticiones Cliente <-> Proveedor

Una vez solucionada esta parte requeriremos de otro ordenador con banda ancha que pueda recibir peticiones y procesarla entre otras cosas.

Algunos de los aspectos principales del cliente de MCloud son:

- Necesidad de una máquina situada en la zona cliente(Congo).
- Ejecuta dos servicios concurrentemente usando *JavaScript* como lenguaje principal.
- Para organizar los distintos tipos de peticiones, consultas y almacenamiento de datos se este sistema usa ***Mongodb***.
- Es aconsejable que tenga al menos dos tarjetas de red si es un entorno grande.
- Se crea un protocolo de comunicación MCloud alrededor de *ftp*.
- En nuestros esquemas mostrados en la [figura 9] se usa el satélite como el medio de conexión de alto coste y bajo rendimiento.

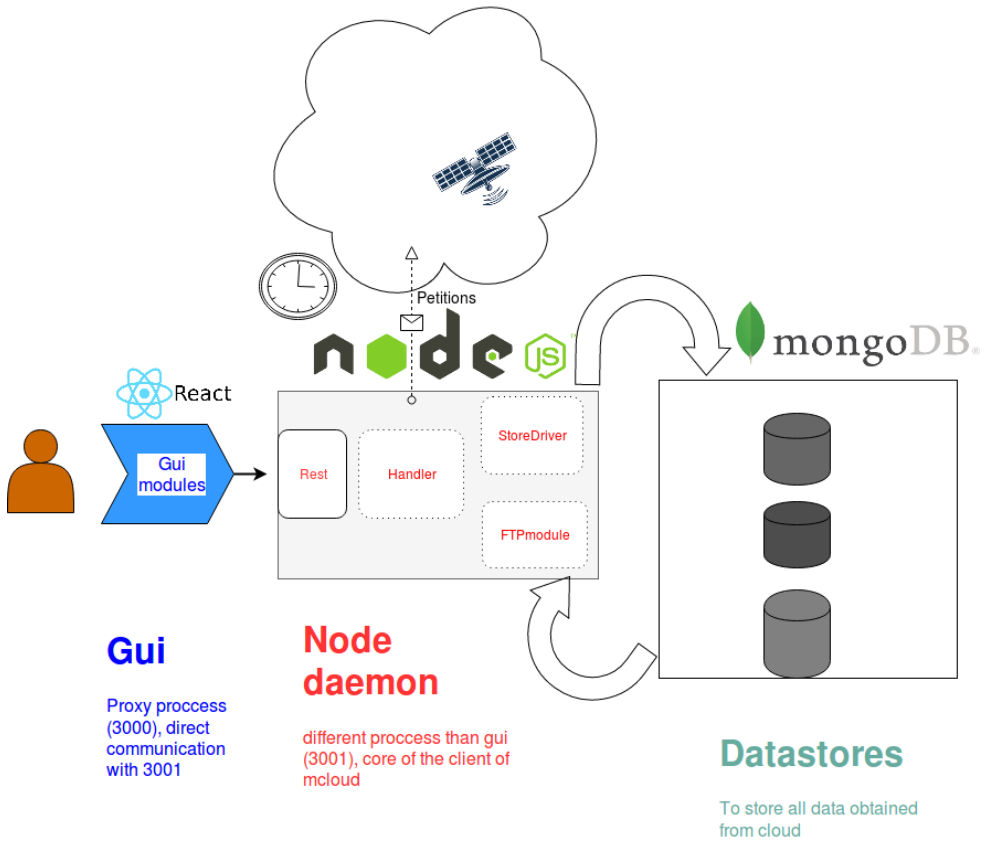


Figura 9: Estructura cliente

3.3.2.1 Arquitectura cliente

Como ya hemos comentado, el cliente esta montado sobre *Frameworks JavaScript* (explicaremos más detalladamente las distintas partes de la zona cliente en las próximas secciones), el siguiente diagrama [figura 10] nos muestra de forma general como funciona el flujo de llamadas y los distintos componentes que lo forman:

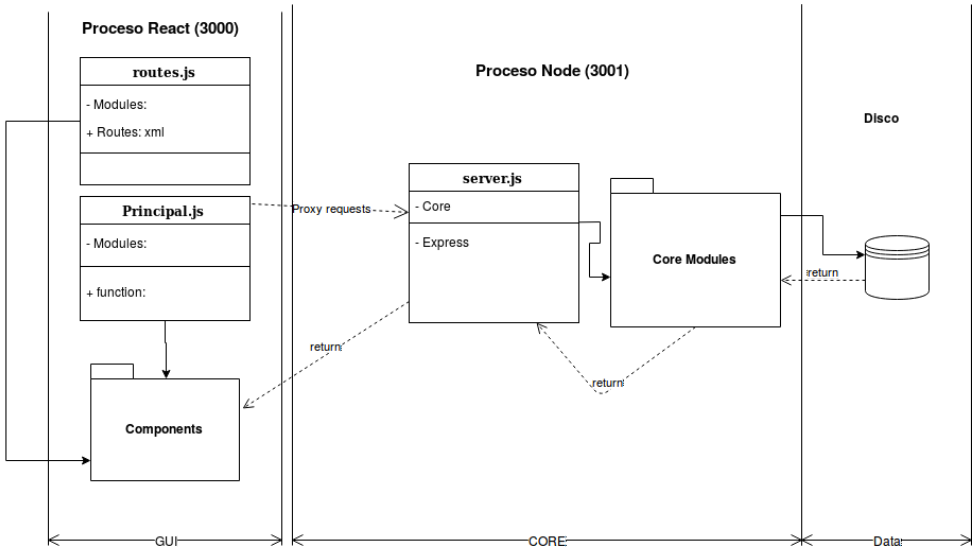


Figura 10: Diagrama clases zona cliente

En este esquema podemos apreciar:

GUI

Proceso (3000) creado con el bundler webpack que actúa como interfaz de la aplicación, los usuarios clientes del sistema cloud podrán lanzar peticiones HTTP (GET) sobre este proceso. En esta parte del código encontramos un router para el proceso 3000 y módulos de interfaz gráfica que desarrollaremos según la necesidad.

Las peticiones recibidas por el puerto *GUI* 3000 son reenviadas de inmediato para el core o aceptadas en caso de que sea una petición de cambiar de vista a el router (routes.js)

Core

Proceso (3001) corriendo el motor *Nodejs* que actúa como *back-end* de la aplicación, recibirá peticiones reenviadas de la GUI o peticiones normales HTTP. El core además de recibir peticiones, tendrá una arquitectura modular con componentes que realizaran la lógica necesaria de la aplicación cloud, pueden ser módulos de cualquier forma (acceso de discos o File system, comunicación, manejo del so...).

Una vez conocido el comportamiento del sistema cliente vamos a explicar sus componentes de forma más específica.

3.3.2.2 GUI

Funcionamiento básico

El front-end de una aplicación suele ser muy característico, en nuestro caso hemos tenido que usar alguna librería externa que se encargue del análisis de código para poder hacer todo correctamente.

Para poder gestionar el enrutado de paquetes, hemos decidido instalar *Browserify*, este paquete nos permite realizar *require* de cualquier modulo que tengamos instalado de forma rápida.

```
    var [nombre] = require('nombreModulo');
    var [nombreComponente] = require('[nombreModulo]').[componente];
```

- Por último, como compilador, es necesario instalar *Babel*, como se mencionó en la *sección 3.1* más concretamente en el apartado de *ReactJS* necesitamos traducir el código JSX a JavaScript para que pueda ser

interpretado por el navegador.

Estructura de la aplicación.

La aplicación corre sobre el puerto 3000, React se encarga de dejarnos una arquitectura que nos permite definir módulos visuales para poder representar la interfaz de la aplicación.

Su desarrollo es imprescindible ya que es la manera que tiene el usuario de mandar instrucciones al core de MCloud, como podemos ver en el diagrama que se nos muestra en la [figura 10], hemos escogido un diseño muy básico que satisfaga a pantallas de baja resolución.

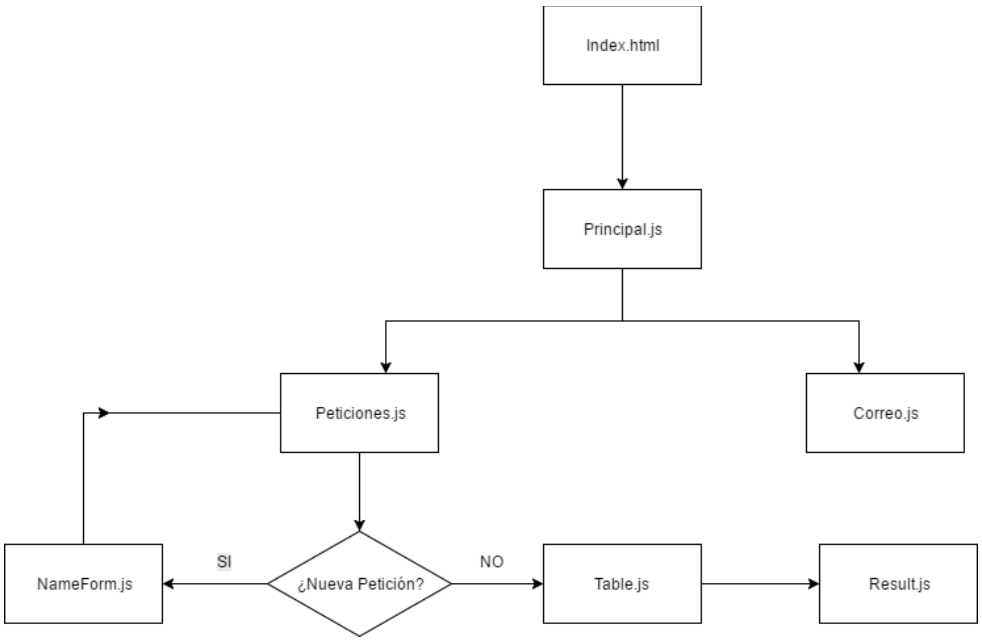


Figura 10: Estructura Zona cliente

- **Index**: Punto de entrada de la aplicación. (SPA)
- **Principal**: Pantalla principal donde se encuentran los botones iniciales.
- **Peticiones**: Pantalla de solicitud de una nueva búsqueda en el sistema o al servidor, esta pantalla esta compuesta de dos componentes:
  1. **NameForm**: Componente donde se encuentra el formulario de petición de nuevas búsquedas.
  2. **Table**: Componente que muestra los datos almacenados en el sistema en referencia al elemento que se esta buscando.
  3. **Result**: Pantalla que muestra el contenido de resultado seleccionado de la tabla.
- **Correo**: Pantalla en la que podemos visualizar la bandeja de correos y enviar uno nuevo.

Componentes externos utilizados

- React-Foundation
- React-Fa
- React-Router
- React-DOM

- React_Notification-System
- Fixed-data-table
- jQuery

3.3.2.2.1 Módulo Peticiones

Este es el componente principal en el cual se representa el formulario para solicitar las nuevas peticiones de búsqueda y una tabla con los resultados (si los hubiera). Como se comentó en la *sección 3.1* una de las ventajas más destacadas es que los componentes que lo forman son reutilizables y muy fáciles de mantener al estar divididos en porciones pequeñas, por este hecho en concreto nos sirve para manejar tanto las peticiones del módulo de *Wikipedia* como las de *YouTube*. Este componente está formado por otros dos *NameForm* y *Table* que veremos más en detalle a lo largo de esta sección.

Métodos.

Constructor

Crea el componente, hereda las propiedades (props) del componente desde el cual es llamado y define los atributos propios del componente.

```
      constructor(props)
```

solicitaDatos

Se encargada de realizar las peticiones (POST) para obtener los datos que el usuario ha solicitado durante su búsqueda, a continuación podremos ver como esta función será pasada al componente *NameForm* que será el encargado de recoger dichos parámetros para poder ser procesados y enviados.

```
     solicitaDatos(data, resource, path, myThis){
    var urlpath = path;
    $.ajax({
        url: urlpath,
        type: "POST",
        data: JSON.stringify(data),
        contentType: "application/json",
        success: function(response) {
          console.log(response);
        },
        error: function(response) {
          console.log("error");
        }
    });
  }
```

handleChange

Este método es el encargado de indicar si hay datos relacionados con los parámetros de búsqueda introducidos por el usuario,diferenciando si la búsqueda se está realizando desde el módulo de *Wikipedia* o de *Youtube* esta función será utilizada en el componente *Table* que hará que la tabla se muestre o no en función de si existen coincidencias entre los datos introducidos y los previamente almacenados.

```

handleChange(event)
```

render

Este es el método que hace visible el componente, puede apreciarse como se llama al componente *NameForm* y el componente *Table* pasándola como atributo los métodos mencionadas anteriormente para que ellos, como componente de forma independiente, realicen las funciones necesarias correspondientes.

```
    render() {
       return (
         <div>
          <form onSubmit={this.handleSubmit}>
           <Row>
            <Col sm={10} large={8}>
             <h1 className="texto_principal_peticiones App-header">
                   {this.name}
             </h1>
            </Col>
           </Row>
          </form>
          <Row>
           <div className="search">
            <fieldset>
             <legend><h3>{"Nueva Peticion"}</h3></legend>
             <NameForm Change={this.handleChange}
                       solicitaDatos={this.solicitaDatos}
                       that={this.thisPrincipal}
                       name={this.name}/>
            </fieldset>
           </div>
           <div className="search">
            <fieldset>
             <legend>Resultados</legend>
             <Table rows={this.table} type={this.name}/>
            </fieldset>
           </div>
          </Row>
        </div>
    );
  }
```

Subcomponente NameForm

Este componente está incluido en el anterior, más concretamente es la parte que corresponde al formulario de búsqueda de nuevas peticiones. Pasamos a analizar los métodos principales que hemos comentado en el apartado anterior.

Métodos

constructor

Método que crea el componente, en este caso las propiedades(props) que recibe el constructor del componente son las que hemos utilizado en el método *render()* de *Peticiones* entre ellos el método *solicitaDatos* que habíamos comentado en el apartado anterior.

```
    constructor(props)
```

handleSubmit

Este método se encarga de recoger los datos introducidos en el formulario cuando el usuario hace click en el botón correspondiente al envío del formulario. Utilizando la función *solicitarDatos* realiza las peticiones, se muestra un mensaje de confirmación en caso de que la petición se haya realizado con éxito o de error si por el contrario, no se ha introducido ningún criterio de búsqueda en el formulario.

```
    handleSubmit(event)
```

render

Pinta el componente teniendo en cuenta, en primer lugar que no se trate del módulo vademécum ya que este no tendrá una opción de búsqueda en el que se especifiquen el número de resultados que se desea recibir. El resto del método muestra el formulario que podemos apreciar en la [figura 12].

```
   render() {
    var numAndSearch = "";
    if (this.name !== "Vademecum") {
      numAndSearch =  <Row className="search">
                        <Column small={5} large={3}>
                          <label>{"Numero de coincidencias"}</label>
                          <input placeholder="3"
                                 ref="number"
                                 type="number"/>
                        </Column>
                        <Column small={2} large={3}>
                          <Button type="submit">
                                 <Send size={Sizes.small} />
                          </Button>
                          <NotificationSystem ref="notificationSystem" />
                        </Column>
                      </Row>;
    }

    return (
      <div className="grid_form_request">
       <form onSubmit={this.handleSubmit}>
        <Row className="search">
          <Column small={12} large={12}>
            <label>search:</label>
            <input
```

```
                placeholder={"introduce la busqueda"}
                onChange={this.props.Change}
                ref="search"
                type="text"/>
          </Column>
        </Row>
        {numAndSearch}
      </form>
    </div>
  );
}
```

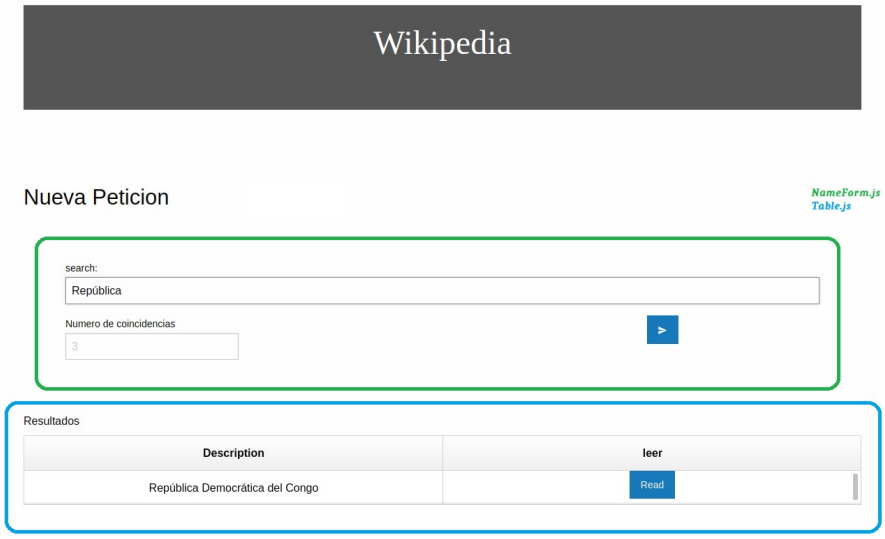


Figura 12: Componente peticiones

Subcomponente Table

Este es el otro componente que se incluye en *Peticiones*. Su finalidad es mostrar la tabla con los resultados de búsquedas que ya se han realizado previamente, mostrando en cada una de las filas los distintos contenidos relacionados con la misma. Cuando pulsemos en alguna de ellas el sistema nos reaccionará al componente *Result* que es encargado de mostrar el contenido.

Métodos

constructor

Crea el componente.

```
    constructor(props)
```

handleSubmit

Cuando seleccionamos una fila de la tabla, este método nos trae el contenido con el que esta relacionado, si todo funciona correctamente se redirige al componente *Result* por medio del *browserHistory.push* añadiendo el salto a esa "ruta" (componente *Result*) y pasándole la información del contenido.

```
    handleSubmit(event) {
    var params = {};
    params.success = function(request,response){
      browserHistory.push({
        pathname: '/result',
        state: { data: response }
      });
    }
    params.error = function(response){
      console.log(response);
    }
    params.data = event.target['id'];
    params.type = "wiki";
    if(params.data !== "")
      CommonActions.list(params, "getData");
    event.preventDefault();
  }
```

render

Esta método muestra la tabla de resultados, como podemos ver en la [Figura 12] en caso de que los haya, si no, en luegar de la tabla simplemente mostrará un texto indicando al usuario que introduzca una nueva búsqueda.

```
    render()
```

3.3.2.2.2 Módulo Resultados

Este es el componente más sencillo de todos, podríamos decir que es el componente mínimo que podemos crear en *ReactJS*, ya que su única funcionalidad es mostrar información, los únicos métodos que tiene son el *constructor* y el *render* para mostrar la información en formato *HTML*.

Métodos

constructor

El componente se crea y recibe una array con los datos que deberá mostrar en su método render.

```
    constructor(props){
    super(props);
    if (Array.isArray(props.location.state.data))
      this.datos = props.location.state.data[0].data;
    else
      this.datos = props.location.state.data.data;
  }
```

render

Este método render simplemente muestra el título del contenido que se está visualizando y a continuación el desarrollo del mismo. En la [figura 14] se refleja la vista del componente, siguiendo con el ejemplo de la [figura 12] del componente peticiones donde se mostraba una tabla con los resultados de la búsqueda.

```
    render() {
    return (
      <div>
        <Row isColumn>
         <h1 className="texto_principal_peticiones">{this.datos.name}</h1>
        </Row>
        <Row isColumn>
          <div className="search">
            <fieldset>
              <legend><h3>Contenido</h3></legend>
                {this.datos.content}
            </fieldset>
          </div>
        </Row>
      </div>
    );
  }
```

República Democrática del Congo

Contenido

La República Democrática del Congo (en francés: République démocratique du Congo, en kikongo: Repubilika ya Kongo Demokratika, en suajili: Jamhuri ya Kidemokrasia ya Kongo, en lingala: Republiki ya Kɔ́ngɔ Demokratiki, en chiluba: Ditunga día Kongu wa Mungalaata), también conocida popularmente como RD Congo, Congo Democrático o Congo-Kinsasa, es un país de África central, denominado Zaire entre los años 1971 y 1997. Situado en la región ecuatorial de África, comprende gran parte de la cuenca del río Congo, extendiéndose hasta la región de los grandes lagos. Es el segundo país más extenso del continente, después de Argelia. Limita con la República Centroafricana y Sudán del Sur al norte, Uganda, Ruanda, Burundi, y Tanzania al este, Zambia y Angola al sur, y la República del Congo al oeste. Tiene acceso al mar a través de una estrecha franja de 37 km de costa, siguiendo el río Congo hasta el golfo de Guinea. El nombre Congo encuentra su origen en los nativos bakongo, asentados en las riberas del río Nzadi o Zaire, rebautizado en portugués como río Congo. La RDC es dueña de una rica y variada historia que se inicia con los primeros inmigrantes bantúes que llegaron a la zona, la cual se convertiría en el epicentro del gran Reino del Congo a mediados del siglo XV. Después de ser reclamado el territorio por la Asociación Internacional Africana (propiedad del rey Leopoldo II de Bélgica) como Estado Libre, y luego tras una colonización particularmente brutal por parte de Bélgica, la colonia del Congo Belga alcanzaría la independencia en 1960, para transformarse en el

Figura 14: Muestra datos wikipedia

3.3.2.3 Core

Estructura

Gracias a nuestro ecosistema *NodeJS* podemos llevar a cabo un desarrollo ligero y directo con un flujo de ejecución controlado.

Las principales características de nuestro *core* son:

Servidor Rest HTTP: server.js

Lo primero que buscamos en el cliente de este sistema es un proceso o servicio que atienda peticiones de los usuarios, lo más sencillo para lograr esto, es el ya bien conocido servidor *HTTP*, estas palabras en *NodeJS* nos llevaron a *Express* una gran librería que nos da justamente eso, los métodos necesarios para tener un servicio de escucha atendiendo peticiones de la manera más ligera posible.

Este entonces, es el primer módulo de nuestro core, a partir de él funciona todo el sistema, este proceso escuchará por el puerto 3001 y se comunicará automáticamente con el puerto 3000 reservado para la GUI.

El módulo consiste en un servicio de escucha http con los métodos para contestar a todas las posibles peticiones, al ser el comienzo de la ejecución del core, se encargará también de crear el resto de módulos además de cargar posibles configuraciones.

Módulo principal:

El módulo principal encargado de la funcionalidad de la parte cliente de MCloud es el manejador de peticiones. Se encargará, como su propio nombre indica ,del manejo de peticiones, gestionar, ordenar, encapsularlas dentro de paquetes... El manejador de peticiones se dedicará a recoger la información brindada por nuestro servidor y convertirla en paquetes que almacenará en el sistema de ficheros de nuestro *SO*.

En él también podremos definir el horario de envío de peticiones de MCloud y se encargará de todas las operaciones con archivos binarios como puede ser la compressión o descompresión de archivos.

Este módulo actúa como una especia de controlador, ya que es el encargado de ejecutar al resto de programas/módulos para MCloud.

Módulo comunicación:

El módulo de comunicación es *ftpwrapper*, es un programa estático que envuelve el protocolo normal File Transmission Protocol y agrega el código de el propio protocolo que usamos en MCloud para la transferencia de peticiones.

Como ya hemos dicho, este se dedicará a hacer llegar las peticiones al proveedor, que tendrá que tener un servidor ftp en escucha para atender las peticiones de este cliente.

Módulo Almacenamiento:

Este sería el último módulo principal de el core del cliente, se encargará de la tarea de almacenar todos los datos obtenidos delegando su almacenamiento final en disco a un servicio *Mongodb* personalizado para MCloud.

Contiene los métodos necesarios para guardar información con *mongodb* y también la de su acceso o modificación.

3.3.2.3.1 Servidor web (HTTP)

Dependencias: * ExpressJS * fs de node

Clase principal del back-end de la aplicación, en ella se encuentra el punto de entrada del motor node, y manejaremos todo lo relacionado con la infraestructura web, ayudándonos para ello de la librería *Express*.

Es responsable de la creación de los demás módulos, pero sobre todo el módulo principal (petitionhandler.js) antes de ponerse a la escucha de peticiones, para ello realiza una carga del código de los demás basándose en un archivo de configuración que se utiliza para parametrizar la parte cliente en función de sus necesidades:

```
const app = express();

var ph = require('./PetitionHandler');
var ftpw = require('./FTPwrapper');
var tools = require('./utils');

var config = require('../../config.json')
console.log(config);

var petition_handler = new ph(config.packet_limit);
ftpw.init(petition_handler, config.provider);


app.set('port', (process.env.PORT || 3001));
app.use(bodyParser.json());

if (process.env.NODE_ENV === 'production') {
  app.use(express.static('client/build'));
}
```

Express nos permite con sus métodos, manejar un servidor que responda a peticiones web, en MCloud hemos creado una interfaz minimalista que reciba las peticiones de la parte *GUI* e interactúe con el *core*. A continuación exponemos un ejemplo de uso que describe a grandes rasgos la manera que tenemos de responder las peticiones por parte del cliente.

Ejemplo de uso

En server.js podemos escribir todo lo relacionado con la web API de una forma sencilla, una forma de hacer que nuestra aplicación respondiera a la petición get de nuestra aplicacion en la ruta /api sería:

```
const express = require('express');
const app = express();

app.get('/api', (req, res) => {
    alert("Ejemplo de llamada");
});
```

req es un objeto que contiene información sobre las peticiones HTTP generadas en ese evento, **res**. Es el objeto que se va a construir en respuesta a esa petición.

A continuación, vamos a enumerar los distintos puntos que tiene el servidor para contestar a las peticiones:

Puntos de entrada del servidor:

Search

Esta función se dedica a interactuar con el módulo principal del core para enviarle una petición de búsqueda.

- Type: POST
- json: `{search : "búsqueda", num : "modulo" }`

```
app.get('/api/search', (req, res, next) => { code });
```

Form

Esta función interactúa con el módulo principal para que envíe una petición y la procese:

- Type: POST
- json: `{search : "búsqueda", num : "modulo" }`

```
app.post('/api/form', (req, res) => { code });
```

LLamadas API TEST

También tenemos entradas para probar las funcionalidades de MCloud o arreglar fallos en el que caso de que ocurrieran, estas distintas llamadas del formato /api/llamada interactúan con las principales funciones del módulo de comunicación: * /api/

```
    * push:  realiza una llamada push del protocolo de comunicación
    * pull: realiza una llamada pull del protocolo de comunicación
    * fetch: realiza una llamada fetch del protocolo de comunicación
    * flush: realiza una llamada flush del protocolo de comunicación
    * generate: genera una batería de peticiones de prueba
```

Son de tipo Get para su acceso a través de URL, sirven para comunicarse con el módulo principal de forma manual.

3.3.2.3.2 Módulo principal

Dependencias:

- modulo Archiver.
- fs de node

Clase de la parte back-end de la aplicación que se encargará de manejar las peticiones que se van a enviar al otro extremo y comprimirlas en paquetes.

Es el módulo principal del core, se encargará de manejar otros módulos al igual de atender a las peticiones que le pase nuestro servidor.

Constructor

Usado para instanciar un objeto manejador de peticiones. El parámetro *package_size* sirve para determinar el tamaño que va a tener en peticiones cada paquete comprimido. packa_size = 3 significa que cada paquete va a tener 3 peticiones.

qoldix y *qnewidx* son índices que nos sirven para saber el estado o el número de elementos de nuestra cola, *qstorage* será la memoria que guarde las que ya están listas en la cola preparadas mientras que *petitions* se encargará de tener guardadas en memoria las peticiones hasta que se alcance el número de paquetes deseado.

La relación del módulo principal con el módulo de almacenamiento es 1:1 por lo tanto sera también el encargado de crearlo y su control.

```
var PH = module.exports = function(package_size){
  Inicializadores:
                  Modelo
                  options
                  _qoldidx
                  _qnewidx
                  _qstorage
                  _package_size
                  _petitionsObj
                  _petitionsNum
};
```

Mecanica Principal del módulo

Crear Paquete

Función privada de la clase que se encarga de generar un paquete con el identificador (id) de la petición pasado por parámetros y la referencia a memoria (petitions) donde se encuentren nuestras peticiones almacenadas.

En su funcionamiento:

- Crea un evento para inyectar código una vez de que el paquete se haya creado:
- Comprueba posibles errores.
- Crea un stream de escritura en el que escribirá dentro de los binarios.
- Cerramos el stream.
- Juntamos todas las peticiones para la compresión.
- Insertamos en la colección fetch del modelo de datos nuestro package id.

```
//this function create a package and enqueue
function create_package(self, type){

  self.emitter.on("newPackage", function(id){
    var archive = archiver("tar",options);

    //good practice to catch this error explicitly
    archive.on("error", function(err) { ... });

    // write stream
    var output = fs.createWriteStream( ... );

    // listen for all archive data to be written
    output.on("close", function() { ... });
    archive.pipe(output);
    self.enqueue(id);

    //Creating the package
    for (var i = 0, len = self._petitionsObj[type].length; i < len; i++){
      archive.append();
```

```
    }
    archive.finalize();
    self.reset(type);
  });
//Data model need to process the data required for fetch operation
self.data.do();
}
```

Recepción de paquete

Función privada de la clase que se encarga de descomprimir todo lo que se encuentre en la ./pull/ y almacenar todos los *JSON* resultantes con los datos en la base de datos. Elimina todos los archivos descargados y descomprimidos cuando estos se han guardado en la base de datos.

En su funcionamiento:

- Compramos el sistema de ficheros (carpeta pull).
- Iteramos sobre los paquetes que encontremos dentro:
- Descomprimíos cada paquete.
- Leemos el contenido de cada paquete:
  - Según cada tipo de peticiones procesamos la información y la metemos al modelo.
- Creamos un observador que se encarga de meter los ids tratados en una lista (remove) para poder realizar la operación flush una vez acabado el proceso.

```
function receive_package(self){
  //we check in pull folder if we have receive_packages to process
  fs.readdir(pull_folder, (err, files) => {

    //we iterate over the files checking the petition type
    files.forEach(file => {

      //tar is in charge to access binary data and call the function to process
      tar.x({ ... }).then(_=> {
         fs.readdirSync(distFolder).forEach(function(file_decompress,index){ ... }
       });

      //Observer dedicated to check pull folder
      self.emitter.on("pulled", function(){
          fs.append(remove) //needed for flush
          self.data.do( ... ); //model operations
      });
    });
  });
```

Funciones

Añadir petición

Esta función se encarga de almacenar una petición en memoria (data) y de crear un paquete en caso de que se haya alcanzado *package_size*, el paquete se encolará para respetar su orden y su tipo ya que al ser MCloud un sistema modular de conocimientos podremos tener distintos tipos de peticiones.

```
PH.prototype.add_petition= function(data){ ... };
```

Seccion Zona horaria

Conjunto privado de objetos y funciones de nuestro moduló principal, se encarga de planificar el horario de comunicación con el otro extremo del sistema y programar trabajos. Puede almacenar objetos time que sirven entre otras cosas para indicar tiempos de activación del sistema o de desactivación. La función de *scheduleJob* es un bloque en el que podemos escribir código que se ejecutará cuando llegue la hora y minutos indicados por el objeto time.

En MCloud gracias a esta interfaz podemos programar que se ejecuten operaciones de tipo push flush o pull a determinadas horas del día, ya que lo único que habría que hacer es llamar a las funciones que activan el módulo FTP.

```
object time
schedule.scheduleJob(time, function(){ ... });
```

search

Función a la cual pasamos un dato a buscar, comprueba en el modelo de datos su existencia y la retorna en caso de éxito. Nos da la opción de pasar un callback para ejecutar codigo junto con la búsqueda. Podemos interactuar con un conjunto de resultados. javascript

```
PH.prototype.search= function(callback, data, type){ ... }
```

reset

Función que se dedica a resetear la cola de peticiones dado un tipo. javascript

```
PH.prototype.reset = function(type){ ... }
```

searchOne

Función que recibe un dato a buscar, comprueba en el modelo de datos su existencia y la retorna en caso de éxito. Nos da la opción de pasar un callback para ejecutar codigo junto con la búsqueda. A diferencia de search en esta función podemos interactuar solo sobre 1 elemento y no un conjunto. javascript

```
PH.prototype.searchOne= function(callback, data, type){ ... }
```

Funciones relacionadas a la cola de peticiones

Size

Nos devuelve el número de elementos.

```
PH.prototype.size = function(){ ... };
```

enqueue

Encola el elemento data, normalmente un identificador para respetar el orden.

```
PH.prototype.enqueue = function(data) { ... };
```

dequeue

Quita un elemento de la cola y lo devuelve.

```
PH.prototype.dequeue = function(){ };
```

Ejemplo de uso

En el siguiente ejemplo vamos a crear un manejador que cree paquetes cada dos peticiones y después vamos a recorrer un bucle guardando "prueba" en cada petición.

```
var n = 2;
var petition_hander = new PH(n);

for (var i = 0, len = 10; i < len; i++){
  var prueba = "prueba";
  petition_hander.add_petition(prueba);

}
```

Resultado:

nPaquetes= len/n = 5 paquetes de dos peticiones, len%n = 0 por lo tanto se han empaquetado todas las peticiones.

3.3.2.3.3 Módulo ftp

Dependencia: modulo FTP.

Funcionalidad de la parte back-end del servidor (node) que se encargará de la parte cliente-servidor relativa al protocolo de comunicaciones necesario para enviar los paquetes con peticiones.

Es el responsable de la implementación de las distintas funcionalidades de nuestro protocolo de comunicacones sobre ftp:

```
var action = {
    idle : 0,
    push : 1,
    pull : 2,
    fetch : 3,
    flush : 4
};
```

Implementación del protocolo de comunicaciones.

Las principales acciones de nuestro protocolo de comunicaciones son: push, fetch, pull, flush. Todas ellas se encargan de hacer llegar datos al otro extremo, básicamente se conectan por ftp al directorio remoto y acceden al sistema de ficheros del proveedor para hacer comprobaciones.

En esta sección se comentara los detalles de implementación que tiene el protocolo de comunicación de Mcloud, los detalles de diseño que tiene este protocolo se detallaron en la seccion 3.3.1 Comunicación.

A continuación pasaremos a describir cada bloque de código por separado:

Push

Este bloque se dedica a iterar sobre todos los paquetes almacenados en la carpeta de push, en cada iteración y a través de un socket FTP envía la información a la zona proveedora.

```
            case action.push:
              fs.readdir(pushFolder, (err, files) => {
                files.forEach(file => {
                  c.put(function(err) { ... });
                });
              });
            break;
```

Flush

Esta operación es la encargada de borrar los paquetes de peticiones que ya tengamos procesados por el proveedor y almacenados en nuestro modelo.

Principalmente lo que hace es iterar sobre la carpeta borrando todo lo que no necesitemos.

```
            case action.flush:
              fs.readdir(flushFolder, (err, files) => {
                files.forEach(file => {
                  c.put(...) { ... });
                });
              });
            break;
```

Pull

La operación pull tiene el papel de descargar del proovedor los paquetes procesados necesarios, una vez recogida toda la información tenemos que decirle a nuestro modelo de datos que se ha efectuado dicha descarga.

```
          case  action.pull:
               if (handler){
                   handler.emitter.on('pull', function(item){
                       c.get(fetchFolder+name , function(err, stream) {
                         stream.once('close', function() { c.end(); });
                         stream.pipe(fs.createWriteStream(pullFolder+name));
                         handler.pull();
                       });
                     }
                   });
                   handler.data.do(...);
                 }
               break;
```

Fetch

Este bloque de código es el encargado de consultar si esta disponible un dato para la descarga, apoyándose en el manejador que consulta el modelo, básicamente consulta en unos datos cacheados los paquetes pendientes de descarga y marca una flag cuando están preparados.

Entre otras cosas también nos sirve para mantener un orden a la hora de descargar todos los paquetes.

```
            case action.fetch:
                c.list(fetchFolder, function(err, list) {
                    list.forEach(function(elem){
                      if (...) {
```

```
                        handler.data.do(...)
                      }
                      else if(...) {
                        handler.data.do(...);
                      }
                    });
                  c.end();
                });
                break;
```

Métodos principales

make_operations

Función que se llamará una vez la petición de **entrada** ftp tenga éxito, en este caso lo único que haría es listar el contenido del sistema de ficheros de entrada de nuestro servidor ftp.

```
function make_operations(list){ ... }
```

Evento ready

c es el objeto sacado de la dependencia ftp donde guardamos la estancia del cliente que accede al servidor. Una vez que la conexión ftp se ha realizado con éxito (cliente->servidor) la librería ftp dispara un evento llamado "ready", esta es la definición de ese evento que ejecutara un callback pasado por parámetro a list(callback), ese es el motivo de que la siguiente función que llamemos sea make_operations (solo se ejecutará una vez se realize la conexión), c.list() recibe como parámetro el objeto list que contiene la lista de archivos/carpetas dentro del sistema de ficheros que se ha accedido

```
c.on('ready', function() { ... });
```

send_info

Esta es la función principal que enviará un objeto con los datos de conexión que requiere el servidor para conectarse tales como la password, el usuario...

```
function send_info(){ ... }
```

3.3.2.3.4 Módulo almacenamiento

Dependencia: Driver mongo js

Este módulo sirve como modelo de nuestro sistema, se encarga de manejar datos y enviárselos al servicio *mongodb* que es en el que delegamos el almacenamiento de información.

Contiene una serie de métodos que nos permiten comunicarnos con estancias mongo que pueden estar en otros servidores y discos.

Funcionamiento

El funcionamiento del módulo de almacenamiento es bastante simple, se dedica a ejecutar código sobre una suite de operaciones y llamadas que define este propio módulo para el core de MCloud:

Do operation

Este método es el encargado de ejecutar operaciones y llamadas en el core de MCloud, a continuación veremos las implementaciones.

```
_Data.prototype.do = function(callback, query, data, emitter, type){

    var default_col = 'fetch';

    if (type)
        default_col =_sys.get(type);

    MongoClient.connect(url, function(err,db){
        assert.equal(null, err);

        // see cursor example above
        var cursor = callback(db, query, data, default_col, emitter);

        db.close();
    } );
}
```

Interfaces de almacenamiento

Operation:

Namespace que almacena los métodos con acceso de escritura en el servicio de almacenamiento, en el encontramos operaciones como insertar, modificar o borrar.

Las cabeceras de las funciones de este namespace siguen siempre el mismo patrón:

- db: La base de datos a la que apunta MCloud.
- query: El objeto hash que contiene las tuplas del criterio de búsqueda.
- {data}: Dependiendo de lo que haga la función especifica, contiene el objeto de datos con el que se va a realizar la operación.
- type: El tipo de modelo con el que interactuamos.
- emitter: Este objeto observador se encarga de avisar al módulo principal de que ha terminado la operación.

Algunas de las funciones principales de este namespace son:

Insert Esta función nos permite insertar un objeto *toInsert* en el *type* modelo que deseemos.

```
    insert: function(db,query, toInsert, type, emitter){ ... }
```

InsertData Función homóloga a insert con la diferencia de que nos permite insertar un conjunto de objetos.

```
    insertData: function(db,query, toInsert, type, emitter){},
```

Update Este método toma por parametro un objeto quera y otro toModify, modificar en el modelo de un tipo los objetos que coincidan con el parametro query y remplazará lo que dicte toModify.

```
    update: function(db, query, toModify, type, emmiter){ ... }
```

Remove Función que dada una query, buscará el objeto que coincida y lo borrará de el modelo.

```
    remove: function(db, query, toModify,type, emmiter){ ... }
```

Content:

Namespace en el que residen los métodos con acceso de lectura a nuestro servicio de almacenamiento. Este conjunto de métodos está dedicado para servir información a la aplicación que lo requiera.

- db: La base de datos a la que apunta MCloud.
- query: El objeto hash que contiene las tuplas del criterio de búsqueda por el que filtraremos
- type: El tipo de modelo con el que interactuamos.
- emitter: Este objeto observador transpasara la información consultada al modulo principal una vez acabada la operación.

Algunas de las funciones principales de este namespace son:

GetAll Nos devuelve todos los objetos almacenados en un modelo.

```
    getAll : function(db, query, data, type, emitter){ ... }
```

GetSome Nos devuelve un conjunto de objetos que coinciden con un criterio query.

```
    getSome : function(db, query, data, type, emitter){ ... }
```

Get Función que nos retorna un solo objeto que coincida con el criterio query.

```
    get : function(db, query, data, type, emitter){ ... }
```

3.3.3 Zona Proveedor

El proveedor es una de las partes principales del sistema, este se encargará de administrar las descargas de todos los módulos disponibles en el sistema y presentarlos al sistema cliente, para que puedan ser descargados posteriormente. El sistema de ejecución que realiza este sistema se puede ver en la [Figura 15], esta ejecución se detallará mas adelante.

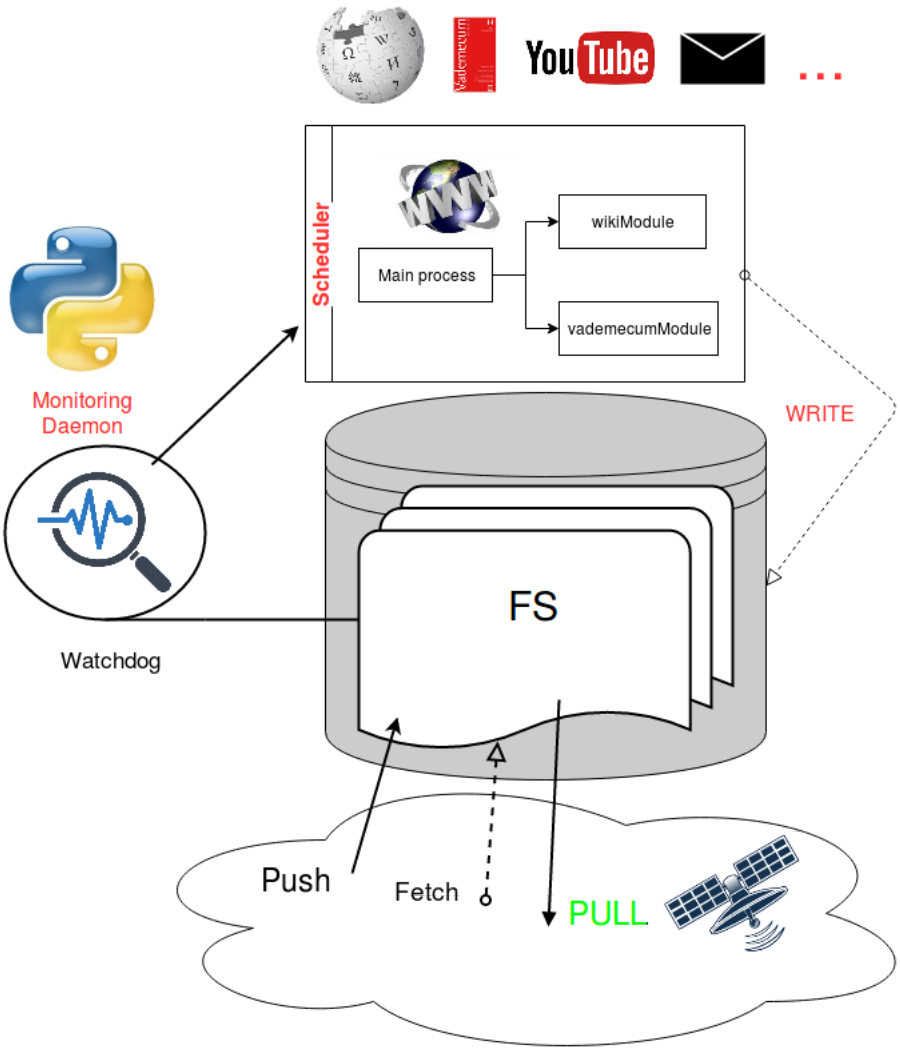


Figura 15: Esquema principal proveedor

Esta zona del sistema ha sido programada en *python*, creemos que este lenguaje de programación facilita mucho a la hora de crear una *API*, en concreto nos ofrecía una amplia variedad de APIs y librerías para ser usadas y su uso era bastante sencillo, por estos motivos nos hemos decantado por este lenguaje.

3.3.3.1 Arquitectura básica

Como podemos ver en la [Figura 16] todo el sistema proveedor está compuesto por una serie de componentes, los cuales realizan una función específica y se comunican entre ellos de la forma mostrada. En próximas secciones detallaremos cada uno de los componentes que encontraremos en el sistema proveedor.

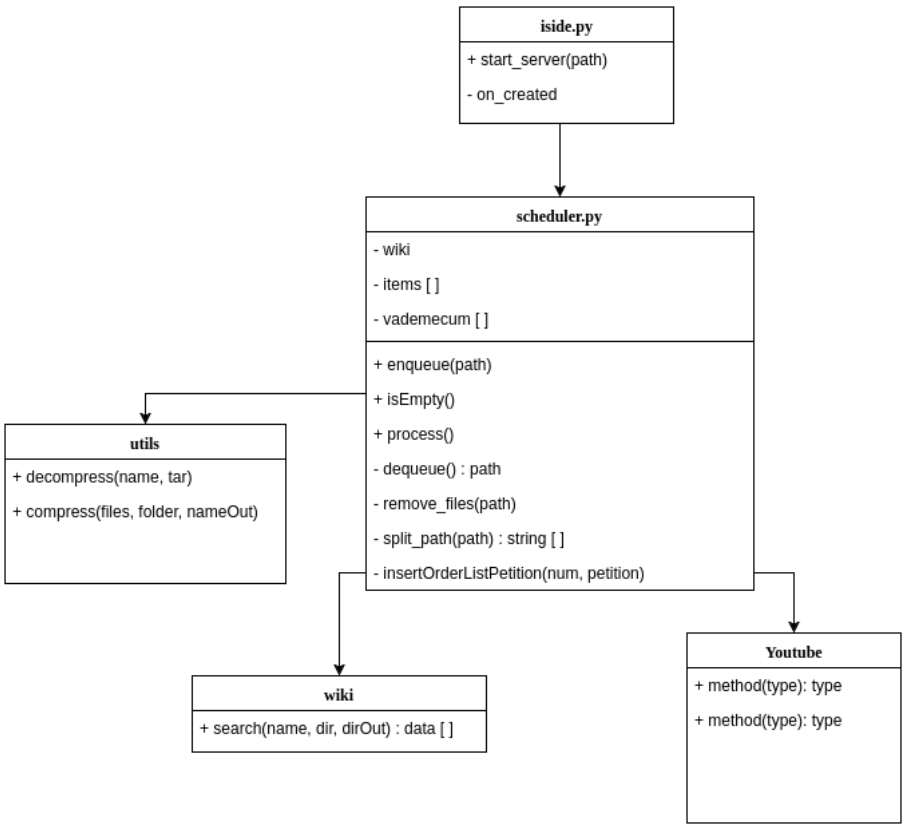


Figura 16: Arquitectura sistema proveedor

Componentes

iside

Este componente es el punto de entrada del sistema en la parte del cliente, este componente se encuentra escuchando en una carpeta `../received/` cuando se produce algún cambio dentro de esta carpeta, el sistema se activa y comunica al componente `scheduler.py` el cambio con el path del archivo introducido.

scheduler

Este componente es el manejador principal del sistema, realiza las tareas de encolado y desencolado de las peticiones de descarga sobre un array de elementos, posteriormente permite procesar estas peticiones de una en una, en este proceso, realiza una serie de comprobaciones y tratamientos sobre el path del archivo, para posteriormente pasar a descomprimir el archivo, solicitar los datos necesarios al módulo especificado en el nombre del archivo y volver a comprimir los resultados en la carpeta `../out/`.

utils

Este componente es simplemente un componente de utilidad que nos permite descomprimir y comprimir archivos.

Wikipedia

El módulo wiki nos permite realizar una búsqueda sobre el *API* `wikipedia` y devuelve los resultados de esa búsqueda.

Youtube

El módulo Youtube realiza una búsqueda a través de unas palabras clave, posteriormente realiza la descarga de los vídeos con esas palabras clave y devuelve los resultados.

Directorio

Todos estos componentes están almacenados en el sistema proveedor con la siguiente estructura:

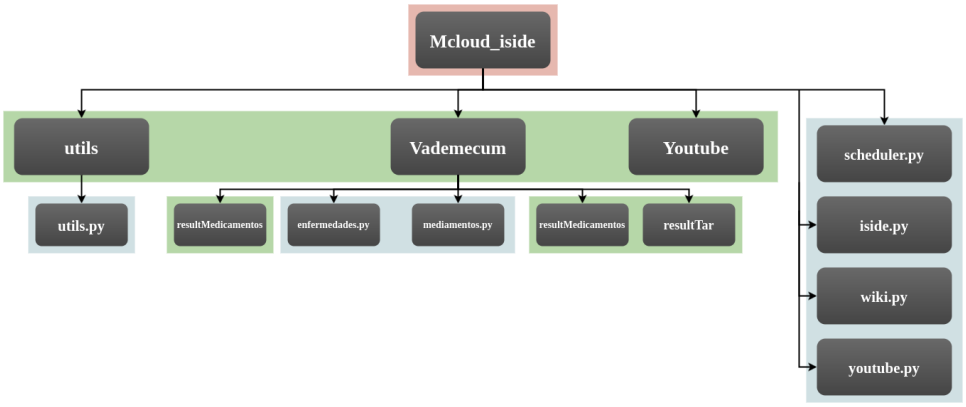


Figura 17: Directorio sistema proveedor

Esta estructura mostrada en la [Figura 17] tiene en su raíz los archivos y módulos principales que realizan la mayor parte de las tareas del sistema, por otro lado, se encuentran las carpetas de utilidades del sistema (`utils/`), la carpeta de almacenamiento temporáneo de los videos que descargará el modulo de youtube (`youtube/`) y como ultima carpeta se encuentra la carpeta del vademécum que ,como se comentará mas adelante, es un módulo con un comportamiento diferente (`vademecum/`).

3.3.3.2 Modelo de ejecución

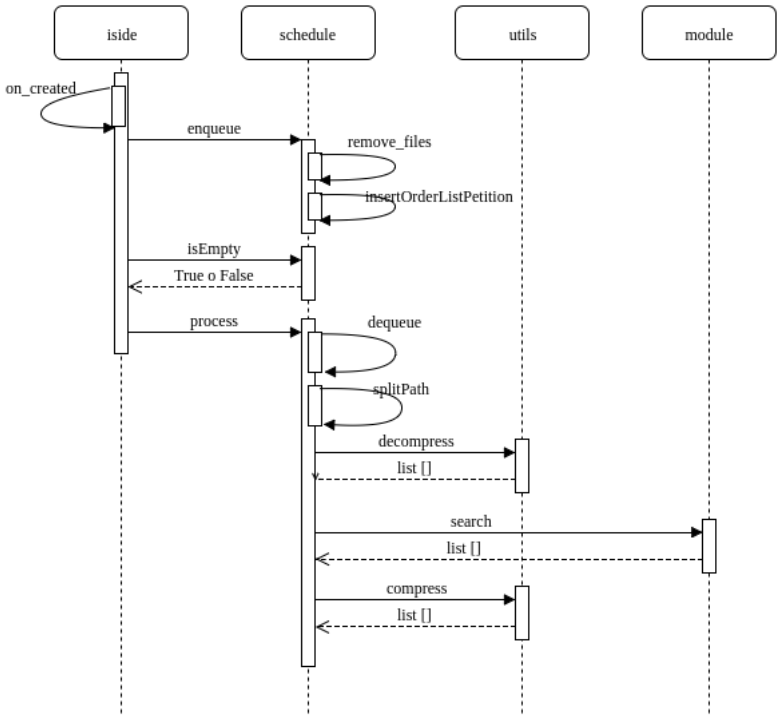


Figura 18: Diagrama ejecución

Este esquema ([Figura 18]) muestra cual es el flujo principal de ejecución de todos los componentes del sistema Provider, muestra las llamadas que realizan estos componentes durante el procesamiento de una petición, desde que un archivo entra en la carpeta `../received/` hasta que se comprimen los archivos finales con la información descargada en la carpeta `../out/`

El proceso de ejecución mostrado en la [Figura 18] sigue las siguientes pautas:

1. El archivo entra en la carpeta de recepción y activa el proceso de encolado de la petición.
2. El sistema de encolado elimina los archivos si es necesario e inserta la petición en orden.
3. De vuelta al sistema principal, se procesa la nueva petición que ha entrado en el sistema de la siguiente forma:

- Se desencola la petición anterior.
- Se procesa y descomprimen los datos de la petición.
- Se realiza la búsqueda de la petición.
- Se comprime la información descargada.

Todas estas funciones mencionadas anteriormente, serán descritas con mas detalle mas adelante en la descripción detallada de cada componente del sistema proveedor.

3.3.3.3 Iside

La clase iside es una de las mas importantes de la zona proveedor de MCloud, en ella esta el punto de entrada de el programa que se encargará de estar escuchando a posibles eventos del protocolo de comunicaciones de Mcloud.

Para poder crear esta especie de servidor, hemos utilizado una librería llamado Watchdog.

WatchDog

Watchdog es un API de Python y una utilidad de Shell para monitorizar los eventos de un sistema de archivos.

Como utilizarlo

Para crear un script sencillo de motorización de directorio simple necesitaremos:

- Crear un instancia de la clase watchdog.observers.Observer para monitorizar cualquier cambio.
- Implementar una subclase de watchdog.events.FileSystemEventHandler o utilizar una ya implementada como watchdog.events.LoggingEventHandler.
- Si queremos un manejador personalizado utilizaremos watchdog.events.PatterMatchingHandler que es una clase que hereda de FileSystemEventHandler y nos proporciona algunos métodos útiles:

1. on_any_event
2. on_created
3. on_modified
4. on_moved
5. on_deleted

Estructura de la clase

Iside esta pensado para que sea un servidor de escucha en el sistema de ficheros, el proceso python se dedicara a escuchar en una ruta esperando a recibir nuevos archivos (en nuestro caso peticiones de MCloud), una vez recibe cualquier tipo de cambio en el sistema de ficheros, Watchdog tiene preparados una serie de eventos que podemos utilizar para activar las funcionalidades:

Interfaz

```
        """
        event.event_type
            'modified' | 'created' | 'moved' | 'deleted'
        event.is_directory
            True | False
        event.src_path
            path/to/observed/file
```

Métodos

Watchdog nos deja esta interfaz de eventos para saber lo que ocurre en nuestro sistema de ficheros, como a nosotros nos interesa saber cuando llegan archivos por ftp, el evento usando principalmente sera **on_created**:

Evento de creación

El funcionamiento no podía ser mas simple, on_created nos indica si hay algún archivo nuevo en nuestra carpeta que no estuviera antes, si es así, lo único debemos hacer, es decirle a nuestra clase principal que encole ese archivo para su proceso posterior.

```
  def on_created(self, event):
        scheduler.enqueue(event.src_path)
```

Inicio del servidor

El servidor se basa en un bucle while true que está constantemente a la escucha con unos observadores de por medio que estan alerta a cualquier cambio, si hay alguna clase de petición en la cola se desencolara para procesarla.

```
def start_server(path):
    observer = Observer()
    observer.schedule(MyHandler(), path, recursive=True)
    observer.start()
    try:
        while True:
            time.sleep(1)
            if not scheduler.isEmpty():
                scheduler.process()
    except KeyboardInterrupt:
            observer.stop()
    observer.join()
```

Chequeando directorio

Esta función se ejecuta antes del inicio del servidor, sólo se dedica a comprobar en el sistema de ficheros si hay algún archivo que no haya podido ser atendido.

```
def check_directory(path):
    print "check_directory"
    print path
    print os.getcwd()
    for filename in os.listdir(path):
        scheduler.enqueue(path+filename)
        scheduler.process()
```

Documentación complementaria.

Documentación sobre la última versión liberada.

3.3.3.4 Manejador

Este componente es el manejador principal del sistema, realiza las tareas de encolado y desencolado de las peticiones de descarga sobre un array de elementos, posteriormente permite procesar estas peticiones de una en una, en este proceso, realiza una serie de comprobaciones y tratamientos sobre el path del archivo, para posteriormente pasar a descomprimir el archivo, solicitar los datos necesarios al modulo especificado en el nombre del archivo y volver a comprimir los resultados en la carpeta ../out/.

Funciones

Process

Función más importante del componente, esta función realiza la descompresión, tratamiento y compresión de todas las peticiones, eliminando los archivos de la carpeta `../received/` una vez han sido utilizados. Llama a un módulo distinto dependiendo del tipo de petición de información que venga marcado en el path del archivo.

```
def process(self, p = None):
        petition = self.dequeue() if p is None else p
        names = self.splitPath(petition)
        try:
            list = decompress(names[0], petition)
        except OSError as osE:
            os.chdir(self.path)
            shutil.rmtree("../out/"+names[0])
            self.process(petition)
        listOut = []
        os.mkdir(list['dir']+"/result/")
        try:
            for l in list['listDir']:
                if names[0].split("_")[1] == "wiki":
                    params = self.wiki.search(...)
                    if params and len(params) > 0:
                        for d in params:
                            listOut.append(d)
                        compress_file = names[0]+"_out.tar.gz"
                        compress(...)
                elif names[0].split("_")[1] == "youtube":
                    self.youtube.search(...)
        except OSError as osE:
            os.chdir(self.path)
            shutil.rmtree(list['dir'])
        except Exception as eEx:
            os.chdir(self.path)
            shutil.rmtree(list['dir'])
            time.sleep(15)
            self.process(petition)
        except e:
            print "cannot get resources, check internet connection!"
            os.chdir(self.path)
            shutil.rmtree(list['dir'])
            time.sleep(15)
            self.process(petition)
```

```
        os.remove(petition);
        os.chdir(self.path)
```

Función process

Enqueue

Encola la petición en el array de peticiones, si el archivo es de eliminación (remove) llama a la eliminación de archivos.

parámetros entrada:

- petition: petición que se va a encolar en el sistema.

```
def enqueue(self, petition):
```

Dequeue

Desencola la ultima petición almacenada en el array de peticiones.

parámetros salida:

- petition: petición desencolada.

```
def dequeue(self):
```

Split Path

Recibe un path de petición y realiza las acciones necesarias para devolver únicamente la información importante del path de la petición.

parámetros entrada:

- petition: petición a tratar.

parámetros salida:

- names[]: array de strings de la petición tratada.

```
def splitPath(self,petition):
```

Remove files

Elimina todos los archivos nombrados en el archivo `remove` de la carpeta `../out/` comprobando siempre que el archivo exista, por ultimo, elimina el archivo `remove` de la carpeta `../received/`.

parámetros entrada:

- petition: petición a eliminar.

```
def remove_files(self, petition):
```

Is Empty

Devuelve True si la lista de peticiones no tiene ningún elemento.

parámetros salida:

- Boolean: booleano de lista vacía

```
def isEmpty(self):
```

Insert Order List Petition

Realiza una inserción en la lista de forma ordenada, esta ordenación depende del tipo de módulo al que vaya destinado, estos módulos tienen una prioridad, la Wikipedia es el módulo que más prioridad tiene, seguido del *Vademécum* y por último el módulo *Youtube*.

parámetros entrada:

- num: numero de prioridad de la petición.
- petition: petición a insertar en orden.

```
    def insertOrderListPetition(self, num, petition):
```

Call Thread

Esta función crea un thread y le manda a ejecutar un comando enviado por el parámetro popenArgs, la llamada `subprocess.call(popenArgs)` se queda bloqueada hasta que el comando ha terminado, cuando el comando ha terminado, llama al callback enviado en el parámetro `onExit`.

parámetos entrada:

- onExit: callback que será llamado cuando el thread acabe.
- popenArgs: argumentos de la llamada al sub-proceso.

parámetros salida:

- thread: thread que se va a ejecutar.

```
    def callThread(self, onExit, popenArgs):
```

Job

Esta función define un callback a ejecutar cuando el comando se termine de ejecutar y manda crear un thread con el comando `scrapy runspider vademecum/medicamentos.py`.

```
    def job(self):
```

Fichero de eliminado (remove)

Este fichero es enviado desde el servidor, contiene una lista de nombres de archivos que deben ser eliminados de la carpeta `../out/`, ya que estos archivos ya se encuentran almacenados en el sistema del servidor.

El formato de este archivo es:

`hashPetition_typePetition`

ejemplo:

`0891w12e2sqd299291_wiki`

3.3.3.5 Módulos

Estos son los módulos que actualmente proveen de información al sistema.

Wikipedia

Este es uno de los módulos de acceso a contenidos que tiene el sistema, en concreto, este módulo permite la descarga de información de la wikipedia, en nuestro caso, debido a las necesidades del proyecto, el módulo esta configurado para realizar las descargas de información en Francés.

Youtube

Este es el módulo encargado de facilitar el acceso a vídeos de la plataforma digital YouTube, este módulo nos permite descargar los vídeos a través de la búsqueda de unas palabras clave, como en el módulo de la wikipedia, podemos decidir el número de vídeos que queremos descargar, este número indicará el número de vídeos en la lista de la búsqueda a los que quieres acceder, descargará los videos en orden.

Vademecum Medicamentos

Este módulo realiza la descarga de los medicamentos que encontramos en la página de Vademecum.es, para no descargar gran cantidad de datos cada vez, se ha decidido descargar por letras del abecedario.

Vademécum Enfermedades

Este módulo realiza la descarga de las enfermedades que encontramos en la pagina de Vademecum.es, como con el otro módulo del vademécum, este también descargará por orden alfabético.

A continuación pasaremos a detallarlos.

3.3.3.5.1 Módulo Wikipedia

Este es uno de los módulos de acceso a contenidos que tiene el sistema, en concreto, este modulo permite la descarga de información de la Wikipedia, en nuestro caso, debido a las necesidades del proyecto, el modulo esta configurado para realizar las descargas de información en Francés, esto se realiza a través de la llamada `wikipedia.set_lang("fr")`. Este módulo únicamente tiene una función a la cual debemos llamar a la función search con los siguientes parámetros: * name : Nombre del archivo a leer. * dir : path del archivo. * dirOut : path de destino donde guardará los resultados.

search

En esta función, el módulo leerá el archivo de entrada, realizará una búsqueda en la Wikipedia del término y por último almacenará los resultados obtenidos en un array de parámetros en los que cada parámetro contiene el nombre de la búsqueda, el contenido de la Wikipedia, la url de la búsqueda y los links asociados.

Modulo externo utilizado: wikipedia API

3.3.3.5.2 Módulo Vademecum medicamentos

Este módulo es distinto a los demás, ya que no es una api a la que llamar para que realize las operaciones, sino que necesita ser llamado como un script a través de la llamada *scrapy*. Para poder ejecutar este módulo se necesita realizar el siguiente comando: `scrapy runspider <fileSider.p>`, esta llamada activará el sistema y se

pondrá a realizar la descarga de los contenidos. Como el Vademecum es un archivo demasiado grande de medicamentos, se ha optado por descargar letra a letra cada vez que se le llame al script, el número de la letra se decide a través de un fichero de configuración llamado `config.txt` este fichero contiene únicamente el número de letra que se debe descargar, aumenta cada vez que se descarga una letra. Para realizar la descarga total de los contenidos por nombres, el sistema consta de dos clases: * Post * MySpider

Clases

MySpider Esta clase, es la clase principal que realiza la mayor parte de la lógica del módulo. Consta de una sección principal en la que se realizan las lecturas y escrituras del fichero `config.txt` y configura el spider.

```
class mySpider(Spider):
    exit = 1
    num = 0
    try:
        print "mySpider ok"
        f = open("config.txt", 'r')
        num = int(f.read())
        f.close()
    except IOError:
        f = open("config.txt", 'w')
        exit = 0
    name = 'mySpider'
    start_urls = ["http://www.vademecum.es/medicamentos-"+ABECEDARIO[num]+"_1"]
    print (num)
    num = num + 1
    if num > 24:
        num = 0
    if exit:
        f = open("config.txt", 'w')
    f.write(str(num))
    f.close()
```

Esta clase también contiene un parseador de las urls obtenidas por el spider, este parseador realiza un parseo de los datos y crea un objeto de la clase Post con los datos que esta necesita.

```
def parse(self, response):
        sel = Selector(response)
        sites = sel.xpath('//ul[@class="no-bullet"]//li')
        items = []

        for site in sites:
            post = Post()
            post['title'] = site.xpath('a/text()').extract()
            post['url'] = site.xpath('a/@href').extract()
            items.append(post)
        for post in items:
            post.download()
        return items
```

Post Esta clase auxiliar del módulo es la encargada de realizar la descarga de los contenidos y almacenarlos en documentos.html, para ello utiliza los datos proporcionados por la funcion `parse` de la clase MySpider. Estos datos son: * url: necesaria para la descarga. * title: nombre del medicamento a descargar, este nombre es el que tomará el documento .hmtl.

```
class Post(Item):
    url = Field()
    title = Field()

    def download(self):
        url = 'http://www.vademecum.es'+str(self['url'][0])
        soup = BeautifulSoup(urlopen(url))
        div = soup.findAll("div", { "class" : "left" })
```

```
        title = str(self['url'][0]).split('/')
        #gs = goslate.Goslate()
        #div = gs.translate(str(div), 'fr')
        f = open("./vademecum/resultsMedicamentos/"+title[1] +".html", 'w')
        f.write("<!DOCTYPE html>\
          <html>\
          <head>\
          <meta http-equiv='Content-Type' content='text/html; charset=utf-8'/>\
          </head>\
          <body>")
        f.write(str(div))
        f.write("</body>\
            </html>")
        f.close()
```

librerías externas

- Scrapy
- Goslate

3.3.3.5.3 Módulo Vademecum enfermedades

El siguiente módulo es el encargado de realizar la extracción de datos de las enfermedades registradas en Vademecum. Debido al escaso ancho de banda sólo se descargará una categoría de enfermedades de Vademecum cada vez que se invoque a la función, de esta manera se irán descargando las diferentes categorías periódicamente. Para hacer esto hemos recurrido a la clasificación alfabética que utiliza Vademecum.

El módulo consta de una función principal llamada enfermedades, que se encarga de toda la carga principal de trabajo y de llamar a las funciones auxiliares necesarias. Esta función obtiene la letra que le corresponda en dicha ejecución, para ello recurre al archivo config.txt. A continuación, obtiene la informacíón de las enfermedades de la página de Vademecum. Y termina con la escritura de esta información en su archivo correspondiente en formato *JSON*.

Funciones auxiliares

- getLetter(): Función que obtiene la letra correspondiente del archivo config.txt, y actualiza ese valor para la siguiente ejecución.
- htmlToFile(entradas): Función que recibe las diferentes paginas de las que obtener la información de las enfermedades, pasa esa información a *JSON* y la escribe en su archivo correspondiente.

```
import goslate
import requests
from bs4 import BeautifulSoup
ABECEDARIO='abcdefghijkmnopqrstuvwxyz'
url = "http://www.vademecum.es"
entrada = "/enfermedades-"
def enfermedades():
    num = getLetter()
    req = requests.get(url+entrada+ABECEDARIO[num]+"_1")
    html = BeautifulSoup(req.text, "html.parser")
    ref = html.find('ul', {'class': 'no-bullet'})
    entradas = ref.find_all('a')
    htmlToFile(entradas)
```

Librerías externas

- Goslate

3.3.3.5.4 Módulo Youtube

Este es el módulo encargado de facilitar el acceso a vídeos de la plataforma digital *YouTube*, este módulo nos permite descargar los vídeos a través de la búsqueda de unas palabras clave, como en el módulo de la wikipedia, podemos decidir el número de vídeos que queremos descargar, este número indicará el número de vídeos en la lista de la búsqueda a los que quieres acceder, descargará los videos en orden. Como el sistema est´s pensado para una conexión de baja velocidad, este módulo descargará los vídeos en una calidad baja, para que estos sean del menor tamaño posible sin perder mucha calidad de vídeo, las calidades escogidas son: * 240p con formato de video 3gp. * 480p con formato de video flv.

Funciones

Search

Parámetros: * name: nombre del archivo a leer en el que se encuentran los términos a buscar. * dir: path donde se encuentra el archivo a leer. * dirOut: path donde almacenará los resultados. * hash: hash de la peticion

Esta función realiza el siguiente proceso, empieza leyendo el archivo proporcionado, a través de este archivo, consigue todos los términos que debe buscar, posteriormente, a través de la url del sistema de búsqueda de la pagina YouTube, recibimos todas las urls de vídeos asociadas a los términos de búsqueda. Seleccionamos sólo las `num` primeras urls de la búsqueda, esta variable `num` nos la proporciona el usuario en la búsqueda. Con todas las urls almacenadas, pasamos a crear objectos del tipo YouTube con cada una de las urls, a través de una sistemas de filtros accedemos a la calidad del video que nosotros buscamos, si no existen las dos calidades de video que nosotros queremos, la función no descargará el video. Por último, almacenaremos los videos en la carpeta indicada.

Finish

Esta función es llamada cuando la descarga ha sido completada, posteriormente realiza la compresión del archivo y almacena este archivo comprimido en la carpeta `out/` .

```
def finish(self, path):
        compress_file = self.hash+"_out.tar.gz"
        compress([self.name],self.dirOut, self.dirFinish+compress_file)
        os.chdir(self.path)
        os.remove(self.dirOut+self.name)
```

Liberías externas

- Pytube

Capítulo 4: Casos de uso

El modelo de casos de uso especifica la funcionalidad que el sistema debe ofrecer desde la perspectiva de los usuarios y cuales son las acciones que los usuarios pueden realizar con el sistema. En este sistema actualmente sólo tenemos un tipo de usuario el cual utilizará la aplicación, este tipo de usuario es un usuario no registrado.

Usuario

Este perfil de usuario es que realizará las acciones principales con la aplicación, estas acciones se muestran en el siguiente modelo.

Modelo

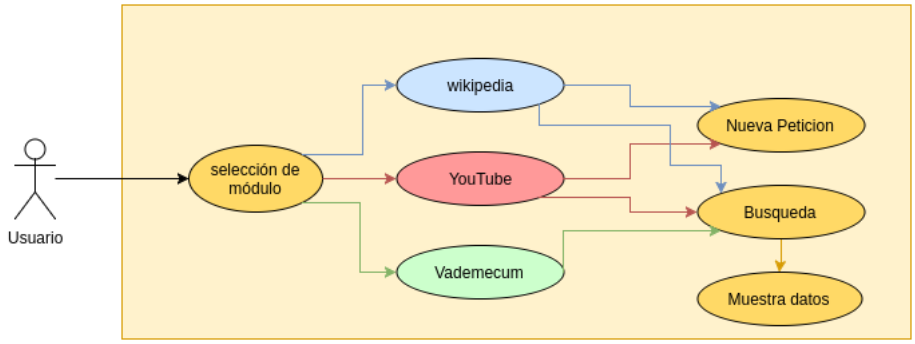


Imagen 16

Ejemplo de uso

Cuando el usuario entra en la aplicación se encuentra con una ventana principal donde se le muestran cuales son los módulos de los cuales puede obtener información, en este momento el usuario puede seleccionar cualquiera de estos módulos simplemente haciendo click sobre los iconos representativos de cada módulo, estos iconos representativos son:

- Wikipedia 𝕎
- YouTube
- Vademecum

El usuario al seleccionar cualquiera de los módulos posibles, pasará a otra ventana donde se le mostrarán una serie de campos a rellenar, el campo principal que se muestra es un campo donde este podrá introducir unos términos de búsqueda, automáticamente, en cuanto que el usuario empiece a teclear cualquier palabra en el campo de búsqueda, el sistema iniciará una búsqueda de esas palabras clave sobre el sistema de almacenamiento y mostrará las coincidencias encontradas en la base de datos en relación a esas palabras, en la tabla de muestra de los resultados se le dará la posibilidad de acceder a la información que ya se encuentra en la base de datos. Si ninguno de las coincidencias encontradas coinciden con lo contenidos que el usuario quiere buscar, este puede realizar la petición de una búsqueda nueva al sistema pulsando sobre el botón con el icono .

Si el usuario pulsa sobre el botón de nueva petición, el sistema registrará su petición y cuando llegue la hora de comunicación con el sistema de Madrid, el sistema enviará la petición empaquetada para que pueda ser procesada y descargada la información.

Cuando el proceso este completado, el usuario no tendrá mas que volver al día siguiente y realizar la misma búsqueda, lo cual le reportará los nuevos datos almacenados en el sistema y podrá acceder a la información.

Capítulo 5: Conclusiones y repercusiones.

5.1 Conclusiones.

El objetivo principal de este proyecto es, tal y como se comentó en la introducción de este documento, proporcionar el acceso a la información y hacer llegar el conocimiento a los lugares que carecen de los recursos necesarios. Este proyecto facilitará la vida a todos los habitantes de la zona, ya sea de forma directa al utilizar los servicios de internet proporcionados por el sistema, como de forma indirecta al beneficiarse de que los profesionales médicos y educativos que ejercen en la zona amplíen sus conocimientos y de esta forma, realicen su trabajo de manera más eficiente y beneficiosa para todos.

Es imprescindible en nuestra condición humana tener acceso a la información para poder aprender, desarrollarnos y aprender sobre lo que nos rodea. Tristemente actualmente este no es un privilegio del que podemos disfrutar todos aunque sea derecho universal, *Mutokamwoyo Cloud* es un proyecto que intentará brindar ese privilegio a toda persona posible sin importar su condición para poder aplicar todos estos conocimientos en áreas básicas para la convivencia y desarrollo humanos.

5.2 Trabajo futuro.

La idea principal era llevar Internet a la zona del Congo más concretamente al pueblo de *Ngandanjika*. La creación de nuestro sistema no es más que el primer paso para llevar a cabo este propósito, la siguiente fase sería viajar al Congo para proceder a la implantación física y el despliegue de nuestro de sistema.

- **1. Implantación en el cibercafé.** El cibercafé es la piedra angular del sistema ya que es el espacio que dispone de los ordenadores y de la conexión directa con el satélite.
- **2. Extensión del servicio al hospital y la zona de la escuela.** Una vez el sistema básico este funcionando correctamente planteamos la posibilidad de extender el servicio de forma inalámbrica por medio de antenas *WiFi* de gran alcance.

5.3 Repercusiones.

Concurso software libre 2017

¿Qué es...?

Es un concurso de desarrollo de software, hardware y documentación técnica libre en el que pueden participar estudiantes universitarios de primer, segundo y tercer ciclo; así como estudiantes de bachillerato, ciclos de grado medio y superior; y universitarios (incluido grado, máster y doctorado) del ámbito estatal español.

texto de CUSL

En este concurso se nos fue entregado el premio al mejor proyecto Cloud.


De Software Libre
Portada
Noticias
Colabora
Proyectos
Bases
Calendario
Comos
Contacto
FAQ
Fase final
CUSL
¿Qué es...?
¿Por qué te tienes que inscribir a este concurso?
¿Qué objetivos perseguimos?
Ultimas Noticias
¿Quieres patrocinar?
Universidades Organizadoras

Chapter 5: Conclusion && Repercussions

5.1 Conclusion

The main goal of this project, as we said in the introduction section of this document, is to provide access to information and be able to carry knowledge to places with limited resources. This project will make inhabitants' lifes much easier, in a directly way by using Internet services provided by our system, or in an indirectly way, because doctors and teachers of the area can use it to improve and extend their abilities making their duties more efficient and profitable.

Is essential in our human being condition to have access to information in order to learn, develop ourselves and know about everything around us. Sadly it is not a privilege for everyone even though nowadays information and internet access are considered as universal rights.

Mutokamwoyo Cloud is a project that tries to offer this privilege to every single person no matter his condition to be able to apply all these knowledge in basic areas in order to improve coexistence and human development.

5.2 Future work.

Be able to carry Internet access to Congo, *Ngandanjika* village to be more specific, was the main idea of this project. Our system creation is just the first step to carry out this purpose, the next phase would be travel to Congo to proceed to the physical implantation and deployment of our system.

- **1. Implantation on the cyber coffee.** This is the leading piece of our system because it is the place that has the computer and satellite connection.
- **2. Spreading the service to the hospital and school area.** Once the basic system will be working properly we have in mind the possibility of spread the service in a wireless way by using range extender *WiFi* antennas.

5.3 Repercussions

Free Software competition 2017

What is..? It is a contest for software development, hardware and free technical documentation in which first, second and third cycle University students can participate; As well as students of high school, cycles of average degree and superior; And university (including degree, masters and doctorate) of the Spanish state.

text of CUSL

In this contest, we won the prize for the best Cloud project.


De Software Libre
Portada
Noticias
Colabora
Proyectos
Bases
Calendario
Comité
Contacto
FAQ
Fase final
CUSL
¿Qué es...?
¿Por qué te tienes que inscribir a este concurso?
¿Qué objetivos perseguimos?
Ultimas Noticias
¿Quieres patrocinar?
Universidades Organizadoras

Capítulo 6: División de trabajo

Sergio Semedi Barranco

Investigación

La primera parte de un sistema software como puede ser MCloud tiene una carga muy pesada de investigación. En este difícil comienzo, Sergio y los demás integrantes del grupo tuvieron que hacer un trabajo conjunto a base de reuniones para decidir la primera parte de diseño inicial y ver como se iba a empezar la investigación.

Sergio después de tener claro el diseño de alto nivel se dedico sobre todo a la parte del cliente, su tarea fue la de investigar a grandes rasgos todo lo relacionado con la tecnología backend de la zona cliente, que en este caso fue Node.

En esta parte sobre todo se dedicó al descubrimiento de nuevas librerías escritas en JavaScript y Nodejs que nos sirvieran para integrarlas en MCloud, este tipo de tecnologías son: librerias FTP, librerías de almacenamiento (mongo), librerías de generación de identificadores...

Una de las partes más importantes además fue la dedicación de tiempo a pensar como hacer que interactúen las partes de MCloud juntas, un ejemplo puede ser la forma en la que el frontend se comunica con el backend a través de un proceso corriendo en otro puerto, Sergio también se encargó de hacer la especificación inicial del protocolo de comunicaciones de MCloud que se tradujo en un modelo muy básico de push y pull ademas de pensar junto a los demás compañeros de equipo la arquitectura inicial del directorio.

Desarrollo

Una vez realizado el trabajo de investigación, Sergio se siguió dedicando en su mayor parte a la Zona cliente y más específicamente el core. Sergio se encargó de la parte de diseño del core además de su implementación más básica, construyo entre otras cosas el modelo basico del core: un servidor web que responda peticiones, un planificador que encolaba peticiones básicas de solo un tipo, un envoltorio sobre FTP para conseguir que funcionara el protocolo de comunicaciones, el módulo de almacenamiento y su especificación inicial, que nos permitiría guardar datos de forma permanente en la aplicación.

En la implementación de todas estas clases hubo un arduo trabajo de ingeniería del software y arquitectura sobre el core, Sergio consiguió poner todo lo que había implementado en funcionamiento.

En la parte de proveedor Sergio colaboro con sus compañeros en los primeros bloques de código que se basaban solo en la primera ejecución de Watchdog y Python sobre el sistema de ficheros del sistema operativo, el objetivo era comprobar el funcionamiento de la comunicación del core con la otra parte del sistema.

Para lograr esta parte del desarrollo, Sergio fue el encargado de la parte de Sistemas, a parte de la ya investigada parte FTP tuvo que hacer las pruebas pertinentes para lograr la configuración adecuada del ya nombrado servidor

FTP y que todo funcionase correctamente en MCloud.

Después de todo esto Sergio se dedicó a colaborar con el resto de sus compañeros en las demás partes del proyecto, colaborando por ejemplo en la intregacion del frontend final-core junto con su sistema de rutas y la resolución de fallos de forma general en todo el proyecto.

Por último Sergio fue el encargado de desarrollar el script de instalación del proyecto completo.

Resumen:

- Encargado principal del core en la parte cliente
- Desarrollo de utilidades para MCloud
- Mantenedor de repositorio
- Resolución de bugs
- Script de instalación

Documentación

En esta parte, Sergio junto con el mantenimiento del repositorio se dedicó a la construcción de la wiki y su diseño inicial. En el inicio de MCloud hubo que decidir la forma de trabajo y el sistema, Sergio se dedicó en esta parte a ser co-mantenedor del repositorio de Github junto con Juan, establecer la arquitectura de ramas y metodología de trabajo además de la redacción de unos tutoriales destinados a todo aquel que quiera colaborar con el proyecto, en ellos se habla de git y de como se trabaja en MCloud. Participo junto con el resto de sus compañeros en construir los documentos iniciales del nacimiento del proyecto (Concurso Mutua, Concurso Software libre...).

Sergio al igual que en desarrollo fue el encargado principal de encargarse de la documentación y explicación de la arquitectura del core, redacto un documento por cada clase que mostrara el uso de los componentes en cuestión.

En la ultima fase del proyecto se dedico a construir esquemas que describieran la arquitectura de la aplicación con su correspondiente redacción de documentos, su ultima tarea fue la escritura de el manual de usuario de la aplicación de MCloud.

Alba Mária Montero Monte

Investigación

Durante la fase inicial del proyecto, Alba al igual que el resto de sus compañeros, tuvieron que reunirse en varias ocasiones para concretar el diseño inicial del proyecto, al ser un proyecto en el que empezamos desde cero, necesitábamos dejar bien definidas tanto las funcionalidades principales como la arquitectura, una vez hecho esto decidimos dividir la investigación en las dos partes principales que hoy conforman nuestro proyecto, el lado del cliente, y el lado del servidor.

También tuvimos una reunión con los miembros de la ONG *Project Ditunga* en la que nos explicaron las características físicas y los medios estructurales de los que disponíasmos, a partir de esto Alba investigó sobre los posibles compoenentes hardware que necesitaríamos para poder cubir las necesidades de la implantación básica del sistema.

Alba junto a su compañero Juan empezaron a investigar la parte correspondiente al front-end, ella se centró en estudiar las posibilidades que ofrecía Angular 2, debido a su popularidad y las funcionalidades que ofrecía gracias a su diseño basado en componentes. Aunque finalmente decidimos optar por ReactJS, (investigado por Juan) ya que la release de Angular 2 habia salido apenas un mes antes y no aún no disponía de mucha documentación para poder trabajar con ella.

Una vez que decidimos que el lenguaje que utilizaríamos sería JavaScript Alba comenzó a investigar Unit Testing (test unitarios) que son pequeñas porciones de código ailados dentro de la aplicación utilizados para asegurar que el código JavaScript de esta funciona correctamente, por ejemplo Jasmine.

También realizó investigaciones por parte de la zona del servidor, más concretamente sobre Python para poder crear un script que monitorizara cambios en una carpeta perteneciente al explorador de archivos, decidió utilizar Watchdog una API de Phyton que proporcionaba las funcionalidades que necesitabamos.

Desarrollo

Después de nuestra fase de investigación, y una vez definidas las tecnologías que ibamos a utilizar en el proyecto, Alba comenzó a implementar funcionalidades dentro de la zona del proveedor, diseñando el script WatchDog que utilizaríamos en la zona cliente para detectar cuando se había recibido una petición nueva, también comenzó a desarrollar la idea del manejador encargado de gestionar las peticiones que llegaban a la zona del servidor aunque finalmente pasó esta tarea a su compañero Juan y ella se dedicó por entero a la parte del desarrollo front-end de la aplicación.

Cuando Alba comenzó con el desarrollo de la parte del front-end de la aplicación su compañero Juan ya había implementado la parte que se encargaba de la comunicación back-end con el lado del cliente, así que ella se dedicó a diseñar toda la parte de la interfaz gráfica diviéndola en los 4 módulos que componen el proyecto (Wikipedia, Vademecum, Youtube y el módulo de correo).

La parte más complicada llegó en el momento del routing para definir el flujo de trabajabo de cada módulo, ya que al tratarse de un modelo de SPA, no existen páginas físicas a las que asignar una url y así poder redirigir el contenido si no que, sobre una misma página dependiendo del contenido, la url tenía que ser distinta.

Para poder llevar a cabo esta fase del desarrollo, Alba decidió instalar varios módulos que proporcionasen las funcionalidades necesarias, en el caso del enrutado react-router fue una pieza fundamental, también se han de destacar los módulos re react-fundation y reac-bootstrap para la creación de tablas y estilos.

Una vez terminado el diseño se encargó de realizar pruebas para comprobar que todo funcionaba correctamente antes de pasar a la fase de documentación.

Documentación

Durante la fase de investigación e implementación Alba recopiló mucha información que después se utilizaría para el desarrollo de la memoria y de la documentación perteneciente a la Wiki del Proyecto.

Alba fue la encargada de gestionar la participación del proyecto en el concurso de software libre, lo cual le hizo recopilar gran cantidad de información y documentación sobre el proyecto ya que uno de los requisitos era el desarrollo de un blog, (https://mutakamwoyocloud.wordpress.com/) en el que se describieran las características principales del proyecto, motivación, objetivos, tecnologías utilizadas...

Para Alba esto fue muy beneficioso a la hora de ponerse a redactar la memoria del proyecto ya que gran parte del trabajo ya estaba realizado.

En cuanto a la memoria, se encargó de toda la documentación relacionada con la tecnología utilizada en el front-end y la explicación de la interfaz gráfica de usuario, ya que es la parte del proyecto a la que ha dedicado la fase de implementación.

Utilizando la información que recopiló anteriormente también realizó la introducción del proyecto, el resumen, las conclusiones, estado del arte, manual de usuario, capturas y diagramas varios, también en colaboración con el resto de compañeros la estructura del documento y aportaciones varias según iban surgiendo necesidades.

Juan José Montiel Cano

Investigación

Durante la fase inicial del proyecto, Juan Jose Montiel Cano se encargó de la investigación de las tecnologías frontend que se encontraban actualmente en el "mercado" y seleccionar cual era la que mejor le venia al proyecto y cuales eran los motivos por los que había escogido la tecnología con la que esta implementada el sistema frontend del cliente.

Durante esta fase se decidió que el lenguaje a utilizar sería Javascript y ,que dentro del gran abanico de librerias y frameworks que están implementados como JavaScript como imagen, en concreto se escogería utilizar ReactJS debeido a su desarrollo en forma de componentes.

Tras escoger este framework, Juan José Montiel Cano, investigó como se podría implementar el diseño de la interfaz web con esta tecnología y como se utilizaba y se programaba en esta tecnología.

Cuando la fase de estudio del lenguaje concluyó, el alumno, organizó cual sería el sistema de componentes que tendría la interfaz web y como se comunicarían entre ellos, también decidió cual seria la forma de la interfaz gráfica y como interactuaría el usuario final con el sistema, es decir, como debería introducir los datos el usuario y como se le presentarían. Para poder completar esta fase de diseño, investigó cuales eran las librerías de ReactJS que mas se adaptaban a los requisitos que ser necesitaban satisfacer.

Otras de las investigaciones que el alumnos realizó, fue la de investigar cual sería el sistema de carpetas de resultados que se producirían en el proveedor y como se generaría los archivos comprimidos finales. En esta parte proveedor, Juan José Montiel Cano, también investigó que módulos y librerías de python existían tanto para hacer webScraping, como para poder descargar video de la plataforma YouTube.

Para terminar con su investigación, ideo una forma para que los archivos ya almacenados en el cliente, fuesen borrados en el proveedor gracias a un archivo de borrado.

Desarrollo

Cuando la fase de investigación inicial estaba ya casi acabando y las pruebas de uso, de las tecnologías a utilizar, dieron buenos resultado, Juan José Montiel Cano empezó el desarrollo del sistema cliente en la parte del frontend, durante estos primeros desarrollos, montó todo el sistema de componente y la interacción que seguirían estos, durante estas fases de desarrollo, también desarrolló la forma de la ventana principal y de los componentes visuales que el usuario utilizaría. Por otro lado, desarrolló un sistema de utilidades para que pudiesen ser utilizadas por todos los componentes, en concreto, la utilidad mas importante fue la del componente CommonActions.js el cual permitía la comunicación con el sistema backend del lado del cliente. Cuando este sistema ya estaba montado, cedió el

testigo a su compañera Alba María Montero Monte, para que esta continuase con la implementación de la interfaz gráfica.

Para continuar con el desarrollo, tras ceder el testigo a su compañera, el alumno se pasó a ayudar y desarrollar en el sistema pero esta vez en la parte del proveedor, en concreto, finalizó el modulo de la wikipedia y comunicó este modulo con el modulo principal del proveedor, también terminó de matizar el compresor y descompresor del sistema proveedor y monto el sistema de carpetas de salida en la parte del proveedor.

Cuando todo el sistema funcionaba con el modulo de l Wikipedia, Juan José Montiel Cano se puso a desarrollar el modulo de YouTube y a probar el funcionamiento del mismo, tras realizar los retoques necesarios para que el sistema y probar que todos los datos se descargaban y se comprimían bien, pasa al modulo de Vademécum, en concreto, el modulo de vademécum de medicamentos ya que el otro modulo de Vademécum estaba siendo desarrollado por su compañero Adrián Martínez Jiménez.

Para finalizar el desarrollo del sistema y poder tocar todas las partes del mismo, el alumno decidió pasarse al desarrollo de algunas utilidades del core (Backend), una de las principales implementaciones en esta parte del sistema, fue la función received_package, que permitía descomprimir y clasificar todos los paquetes comprimidos según el tipo de petición, y posteriormente guardarlos en la base de datos. Tambíen desarrollo ciertas funcionalidades del core e implemento el sistema de comunicación de borrado de archivos entre la parte cliente y la parte proveedor.

Documentación

Durante todos los procesos anteriormente descritos, el alumno Juan José Montiel Cano documento todo lo relacionado con las tecnologías investigas y utilizadas por el sistema en la Wiki del proyecto en Github, esta documentación consistía en comentar todo lo que se iba desarrollando de forma que los demás compañeros si querían introducirse en esa parte del sistema, únicamente tendría que mirar la wiki y ver como estaba montado el mismo.

Fuera de la documentación de la wiki de github, el alumno también documento parte de la memoria que fue entregada para los concursos a los cuales se presento el proyecto, esto significaba que tuvo que crear diagramas, buscar imágenes y recursos utilizados para la creación del documento.

En la fase final del proyecto, el alumno ha documentado todo el sistema proveedor para que este fuese introducido en la memoria del proyecto, también a aportado en otras secciones como el estado del arte, la bibliografía, el sistema de figuras e imágenes, etc...

En esta creación de la memoria final del tfg, el alumno investigó cual seria la mejor forma de creación del pdf final y ,junto con su compañero Sergio Alfonso Semedi Barranco, decidieron utilizar Pandoc, esta herramienta nos permitía crear el pdf a través de un css, este css fue buscado y modificado por Juan José Montiel Cano, el cual añadió ciertas etiquetas css para ser utilizadas en la documentación.

Adrián Martínez Jiménez

Investigación

En esta primera etapa del proyecto, Adrián Martínez Jiménez y sus compañeros tuvieron que ver las dificultades del proyecto así como su desarrollo. Teniendo una idea en alto nivel de como era, realizaron una serie de reuniones con el fin de repartir el trabajo de investigación y tomar las primeras decisiones de diseño.

Una vez escogidas las partes principales del diseño, Adrián Martínez Jiménez se encargó de la investigación de la parte de los módulos y del uso de las API REST de los servicios que debía utilizar la aplicación. Una vez investigadas las diferentes tecnologías y viendo las otras tecnologías utilizadas en las diferentes partes del proyecto, Adrián Martínez Jiménez y sus compañeros tomaron la decisión de utilizar Python para los módulos de los servicios de la aplicación.

Tomando como referencia este lenguaje y los servicios necesarios, Adrián Martínez Jiménez se dedicó a investigar como utilizar las APIS de Wikipedia y Youtube mediante la librería urllib de Python. Posteriormente Adrián Martínez Jiménez encontró diferentes librerías de Python que implementaban estos servicios de Wikipedia y Youtube. Tomando varias como referencia, se dedicó a investigar como utilizarlas y si cubrían los servicios necesarios de la aplicación. Adrián Martínez eligió MediaWiki API como librería para implementar el módulo de la Wikipedia y junto con Juan José Montiel Cano elegieron la librería PyTube para implementar el módulo de Youtube.

Para la parte de Vademecum, Adrián Martínez Jiménez investigó acerca de las tecnologías de web-scraping disponibles para Python y su usó para implementar el módulo.

Debido a la necesidad de mostrarse todos los resultados en francés, Adrián Martínez Jiménez busco información acerca de librerías en Python para traducir textos a este idioma. Tomó Goslate como librería a usar para las traducciones en los diferentes módulos.

Desarrollo

Tras la fase de investigación y teniendo claro las funcionalidades a implementar, Adrián Martínez Jiménez se encargó de implementar los módulos en Python. Empezó construyendo el módulo de la Wikipedia tratando de resolver los problemas de desambiguación de las búsquedas. Adrián Martínez Jiménez consiguió construir el módulo y añadirlo a la aplicación. Continuó con la implementación del módulo Vademecum enfermedades, así como el diseño de las respuestas del módulo Wikipedia.

Adrián Martínez Jiménez ayudó a Juan José Montiel Cano a desarrollar el módulo de Youtube, así como el diseño del compresor y descompresor de la parte proveedor.

A continuación, Adrián Martínez Jiménez introdujo las implementaciones necesarias en los diferentes módulos para que devuelvan sus resultados en francés.

En último lugar, Adrián Martínez Jiménez ayudó a sus compañeros resolviendo problemas del resto del proyecto y probando las diferentes funcionalidades del sistema.

Documentación

Para la parte de la documentación, Adrián Martínez Jiménez trabajó junto a sus compañeros en los documentos iniciales del proyecto que fueron necesarios para presentarse a los diferentes concursos.

Tras esto, Adrián Martínez Jiménez documentó los diferentes módulos de Python. Se dedicó principalmente al módulo de Vademecum enfermedades.

Adrián Martínez Jiménez se encargó de la documentación de la parte hardware y de la intranet, contruyendo los diferenets esquemas y la visión de la investigación anterior.

En último lugar, Adrián Martínez Jiménez corrigió fallos de la documentación.

Glosario de Términos.

Framework:

Un framework, (marco de trabajo) o entorno de trabajo es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

SPA:

Single Page Aplication(aplicaciones de una sola página). Como su propio nombre indica es una aplicación en la que todo el contenido y las vistas son cargadas en la misma página, permiten mejorar la experiencia de usuario proporcionando un mantenimiento más sencillo y una reducción significativa en cuanto al almacenamiento de datos.

SO:

Sistema Operativo. Es el software principal o conjunto de programas de un sistema informático que gestiona los recursos de hardware y provee servicios a los programas de aplicación de software,

Cloud Computing:

Computación en la nube. Es un paradigma que permite ofrecer servicios de computación a través de la red, almacenando información en servidores repartidos por todo el mundo.

IaaS:

Infrastructure as a service(infraestructura como servicio). Es el modelo básico de cloud, representa la capa sobre la cual se sostienen PaaS y SaaS, proporciona un medio de entregar almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red.

PaaS:

Platform as a service(plataforma como servicio). Es la capa que sigue al modelo IaaS, proporciona una plataforma y un entorno que permiten a los desarrolladores crear aplicaciones y servicios que funcionen a través de internet utilizando herramientas suministradas por el proveedor.

SaaS:

Software as a service (software como servicio). Se encuentra en la capa más alta y caracteriza una aplicación completa ofrecida como un servicio, que corre en la infraestructura del proveedor y sirve a múltiples organizaciones de clientes.

API:

Aplication Programming Interface (interfaz de programación de aplicaciones). Es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

JSON:

JavaScript Object Notacion es un formato ligero para el intercambio de datos estructurados, que surgió como alternativa de XML, y que actualmente es uno de los más utilizados ya que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

DOM:

Document Object Model (modelo de objetos del documento),proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML,es un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

GUI:

Graphic User Interface (interfaz gráfica de usuario), es una infterfaz que proporciona un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles, proporciona un entorno visual sencillo para permitir la comunicación con el sistema operativo.

Front-end:

Es la parte del software que interactúa con el o los usuarios, se encarga de recoger los datos introducidos por los usuarios y los transforma ajustandose a las especificaciones del back-end para poder procesarlos y enviar en consecuencia una respuesta al usuario.

Back-end:

Es la parte que procesa la entrada desde el front-end, se encuentra en el lado del servidor y se encarga de la gestión de los datos que recibe del front-end procesándola, almacenandola y transformándola para ser devuelta al front-end y así comunicarla.

Zona Cliente:

Referida a la parte del sistema informatico MCloud a la que se van a dedicar los recursos. En el contexto de el TFG y donde se va a implantar el proyecto esta zona se refiere al sistema informatico que reside en Congo.

Core:

Proceso interno del software de la zona cliente que se encarga de manejar las peticiones, escrito con Nodejs.

Protocolo Comunicación Mcloud:

Referido al conjunto de reglas aplicado que usa el sistema Mcloud para comunicar las dos zonas.

Bibliografía

Estado del arte:

1. *Free Basics by Facebook - Wikipedia*: https://es.wikipedi.org/wiki/Free_Basics_by_Facebook
2. *Free Basics de Facebook – Spanish*: https://info.internet.org/es/stry/free-basics-from-internet-org/
3. _Outernet: https://www.mdif.org/outernet-providing-information-to-the-world-from-outer-space
4. *Project Loon (by Google) - Wikipedia*: https://es.wikipedia.org/wiki/Google_Loon

Desarrollo

Client

Frontend

5. *ReactJS*: https://facebook.gitub.io/react/docs/hello-world.html
6. *FixedDataTable*: https://facebook.github.io/fixd-data-table/getting-started.html
7. *React-Bootstrap*: https://react-bootstrap.github.io/
8. *React-Foundation*: https://react.foundation/
9. *React-Router*: https://reacttraining.com/ract-router/web/guides/quick-start

Backend

10. *Node.js F.*: https://nodejs.org/es/docs/
11. *ExpressJS*: http://epressjs.com/es/guide/routing.html

Provider

12. *Scrapy 1.4.0*: https://docs.scrapy.org/en/latest/
13. *wikipedia 1.4.0*: https//pypi.python.org/pypi/wikipedia/
14. *pytube 6.2.2*: https://github.com/nficano/pytube
15. *watchdog 0.8.3*: htts://pypi.python.org/pypi/watchdog

Gestion de paquetes:

16. *NPM*: https://www.npmjs.com/
17. *PIP*: https://pypi.python.org/pypi/pip

Imágenes

1. https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Escudo_de_la_Universidad_Complutense_de_Madrid_Escudo_de_la_Universidad_Complutense_de_Madrid.svg.png
2. http://mutokamwoyo.org/wp-content/uploads/2014/09/mutokaMowoyo-Logo_v1-e1411895212905.png
3. http://2ctksw85dcap3jo52b29rxwf.wpengine.netdna-cdn.com/wp-content/uploads/2015/12/family-money-charity.jpg
4. http://img01.ibnlive.in/ibnlive/uploads/2015/12/free-basics-vs-say-no-to-free-basics.jpg
5. https://www.mdif.org/outernet-providing-information-to-the-world-from-outer-space
6. https://github.com/MutakamwoyoCloud/DocsMCloud/blob/master/images/projectLoon.jpg
7. https://www.javatpoint.com/images/javascript/javascript_logo.png
8. http://enriquev.github.io/images/react.png
9. http://thisdavej.com/wp-content/uploads/2016/02/nodejs-logo.png
10. http://computerfloss.com/wp-content/uploads/2012/05/python-logo.png
11. http://arunoda.me/images/blog/npm-love-github-thumb.png
12. http://asir.maristak.com/wp-content/uploads/2016/12/unnamed.png
13. https://www.percona.com/sites/default/files/json-logo.png
14. http://i.imgur.com/15qu4FC.png
15. http://i.imgur.com/HLYUJVc.png
16. http://i.imgur.com/CvVnuX3.png
17. http://i.imgur.com/0nqjneh.png

Anexo I: Manual de Instalación

Requerimientos

Zona proveedor:

- Mínimo
  - Maquina con Linux como sistema operativo
  - Servicio FTP
  - Python 2.7.8
  - Pip
  - Conexión a internet estable
  - Direccion ip publica estática
- Recomendado
  - Ubuntu 16.04 Virtualizado
  - Conexión Fibra óptica
  - Velocidad simétrica de más de 10 mb/s

Zona Cliente:

- Mínimo
  - Maquina con Windows o Linux
  - Acceso a la red de alto coste
  - Acceso a la red bajo coste
  - Espacio de disco (el que desee según necesidades)
  - Node y npm instalados

Zona cliente

Dependencias

Instalar o tener pre-instalados los siguientes paquetes:

Npm y NodeJs Versión 7.0+

Lo utilizaremos tanto para realizar la conexión y la transmisión de datos entre las maquinas del Congo y Madrid a través del satélite.

Mongodb Versión 3.0+

```
$ sudo apt-get install npm
$ sudo apt-get install node
$ sudo apt-get install mongodb
```

Consideraciones: _Todas las librerías que utilicemos, deberán ser instaladas y compiladas en el sistema, de tal forma que no necesiten conexión a internet para funcionar.

Una vez tenemos las dependencias resueltas procedemos a descargar o clonar el repositorio:

- Para la descarga directa haz click aqui
- Puedes clonar el repositorio tu mismo: $git clone https://github.com/MutakamwoyoCloud/MCloud.git

Instalando la zona cliente

Dentro del repositorio viene un instalador que nos sirve para ambas zonas, esta en la raíz del repositorio y su nombre es install.sh.

A continuación, explicaremos su uso:

```
Uso: ${0##*/} [-hvd] [-t TYPE]
Instala Mcloud de diferentes maneras


    -h          muestra la ayuda
    -t TYPE     elige que parte de MCloud vas a instalar
    -v          modo verbose
    -d          modo desarrollador

    TIPO -t OPCIONES:

    client:     instala zona cliente
    provider:   instala zona proveedor
```

Ejecutamos el siguiente comando (vease [Figura AI.1]):

```
$ sudo ./install.sh -t client
```

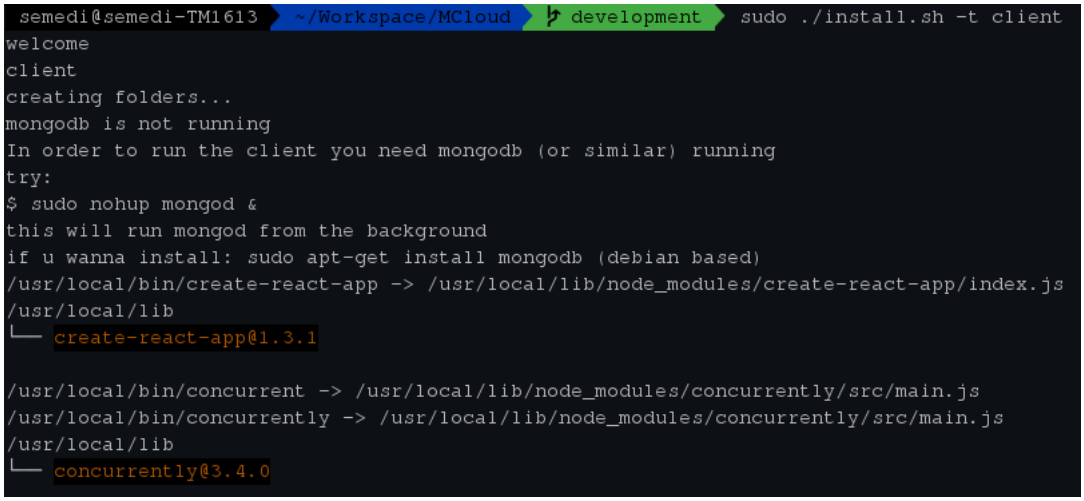


Figura AI.1

Una vez tenemos todas las dependencias y la instalación es hora de ponerlo en marcha, inciamos la aplicación:

```
$ sudo mongod &
$ npm start
```

Figura AI.2

La aplicación cliente se encuentra corriendo como podemos ver en la [Figura AI.2], a partir de este momento los usuarios se podrán conectar a la aplicación a través de navegador indicando URL: puerto.

Zona Proveedor

Dependencias

Instalar o tener pre-instalados los siguientes paquetes:

Python2.7 y pip Version 9.0+

Lo utilizaremos tanto para realizar la conexion y la transmision de datos entre las maquinas del Congo y Madrid a traves del satelite.

Servidor FTP

Para poner Mcloud en funcionamiento tenemos que tener activo en la zona proveedor un servidor ftp que se encarge de atender a las peticiones.

Da igual el software que uses en este apartado siempre y cuando consigas un servidor ftp atendiendo, nosotros en MCloud recomendamos el uso de proftpd, por su facilidad de configuracion:

```
$ sudo apt-get install proftpd
```

A continuacion mostramos una configuracion muy basica (vease [Figura AI.3]) para el funcionamiento adecuado con MCloud:

```
# A basic anonymous configuration, no upload directories.

<Anonymous ~ftp>
User                ftp
Group               nogroup
UserAlias           anonymous ftp
DirFakeUser on ftp
DirFakeGroup on ftp
RequireValidShell       off
MaxClients          10
DisplayLogin            welcome.msg
DisplayChdir        .message
<Directory *>
<Limit WRITE>
DenyAll
</Limit>
</Directory>
</Anonymous>
```

Figura AI.3

Instalando la zona proveedor

Al igual que con la zona cliente, podemos instalar todo lo necesario para el proveedor con el mismo repositorio y el mismo script, esta vez utilizaremos otra opción diferente y elegiremos el tipo proveedor (vease [Figura AI.4]):

```
sudo ./install.sh -t provider
```

```
# A basic anonymous configuration, no upload directories.

<Anonymous ~ftp>
User                ftp
Group               nogroup
UserAlias           anonymous ftp
DirFakeUser on ftp
DirFakeGroup on ftp
RequireValidShell       off
MaxClients          10
DisplayLogin            welcome.msg
DisplayChdir        .message
<Directory *>
<Limit WRITE>
DenyAll
</Limit>
</Directory>
</Anonymous>
```

Figura AI.4

Una vez instalado ya podemos iniciar nuestro servicio para que atienda a las peticiones, nos metemos con el usuario propietario del servicio (por defecto mcloud) y ejecutamos el servidor (vease [Figura AI.5]):

```
$ su mcoud
$ cd ~/MCloud/inet_side/mcloud_iside/
$ python iside.py
```

```
mcloud@semedi-TM1613:~$ ls
MCloud
mcloud@semedi-TM1613:~$ cd MCloud/inet_side/mcloud_iside/
mcloud@semedi-TM1613:~/MCloud/inet_side/mcloud_iside$ ls
__init__.py  Scheduler.py   utils      wiki.py   youtube     youtube.pyc
iside.py     Scheduler.pyc  vademecum  wiki.pyc  youtube.py
mcloud@semedi-TM1613:~/MCloud/inet_side/mcloud_iside$ python iside.py
listening petitions on ../received/
check_directory
../received/
/home/mcloud/MCloud/inet_side/mcloud_iside
```

Figura AI.5

Configuración

Mcloud es configurable, todo el proceso que hemos explicado está hecho siguiendo la configuración por defecto creada para el modo desarrollo. Si quieres realizar otro tipo de instalación es conveniente que edites Mcloud/config.js:

```
{
    "provider" : {
        "url" : "localhost",
        "port" : 21,
        "user" : "mcloud",
        "password" : "mcloud",
        "pull" : "MCloud/inet_side/out/",
        "push" : "MCloud/inet_side/received/"
    },

    "clk" : {
        "on" : {
            "hour" : 20,
            "minute" : 0
        },
        "off" : {
            "hour" : 4,
            "minute" : 0
        }
    },

    "packet_limit" : 2,
    "pullFolder" : "./src/core/pull/",
    "pushFolder" : "./src/core/push/"
}
```

Anexo II: Manual de usuario.

Dado que el uso de nuestra aplicación va dedicado a un grupo de usuarios que no tienen mucha relación con la tecnología optamos por un diseño de interfaz de usuario que fuera muy simple e intuitivo, donde las funcionalidades básicas que pretendemos proporcionar sean sencillas y la navegación a través de la aplicación sea rápida de manera que no sea necesario tener un amplio conocimiento de la misma.

A continuación pasamos a describir el flujo completo para realizar una petición al módulo Wikipedia.

Primero entramos en la página principal de la aplicación.

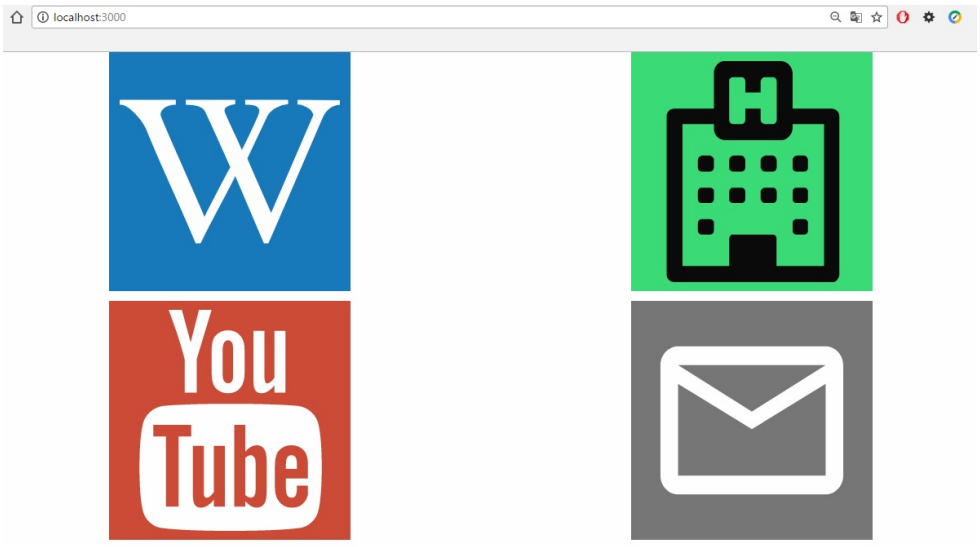


Figura AII.1

Seleccionamos el icono que corresponde a la Wikipedia y acto seguido nos redirigirá a la página que contiene el formulario para solicitar las peticiones del contenido.

Una vez en la página del formulario simplemente tendremos que escribir en el campo search la información que queramos obtener, teniendo la posibilidad de elegir el número de resultados distintos que queremos que se descarguen (por defecto 1) introduciéndolo en el campo proporcionado.

A medida que escribirmos en el campo de búsqueda, si existen resultados relacionados apaparecerá en la parte inferior de la página una tabla mostrando dichos contenidos [Figura AII.3], si no es el caso, simplemente basta con pulsar el botón de envío para que se procese la petición, cuando esto ocurra, se mostrará un mensaje de confirmación al usuario.

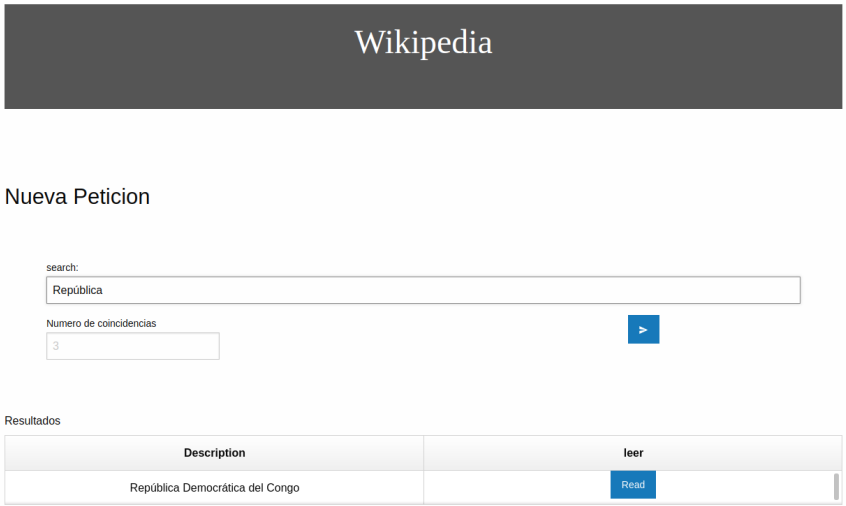


Figura AII.2

Junto a cada fila de la tabla de resultados relacionados que contiene la descripción encontramos un botón "leer", si el usuario lo pulsa será redirigido a la página que contiene el resultado seleccionado.

Figura AII.3

Como el funcionamiento de los módulos Youtube y Vademecum es prácticamente el mismo, vamos a pasar a decribir el módulo correo, ya que su funcionamiento en nada se asemeja con el resto.

Partiendo otra vez de la pantalla principal en la que se muestran los cuatro módulos [Figura AII.1] esta vez pulsaremos sobre el icono del módulo de correo.

Una vez en el módulo, lo que se nos presenta, es la bandeja de entrada de correos del usuario. [Figura AII.5].

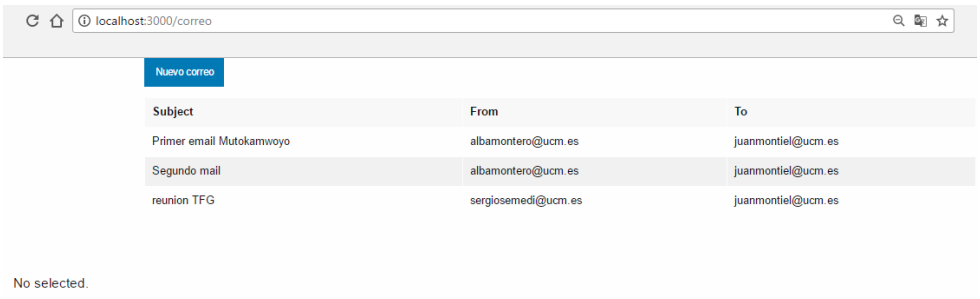


Figura AII.4

Si seleccionamos un correo de la lista,por ejemplo el primero, aparecerá en la parte inferior el correo seleccionado junto con los datos del del emisor, receptor y asunto.

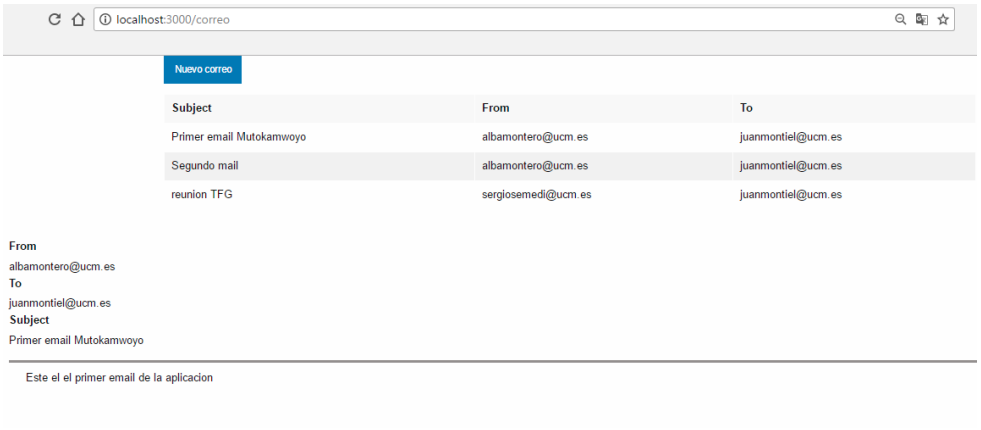


Figura AII.5

Si observamos la parte superior de la página, justo encima de la tabla de correos recibidos, encontramos un botón que pone "Nuevo correo", si lo pulsamos, aparecerá en la parte inferior de la página un formulario para enviar un nuevo correo.

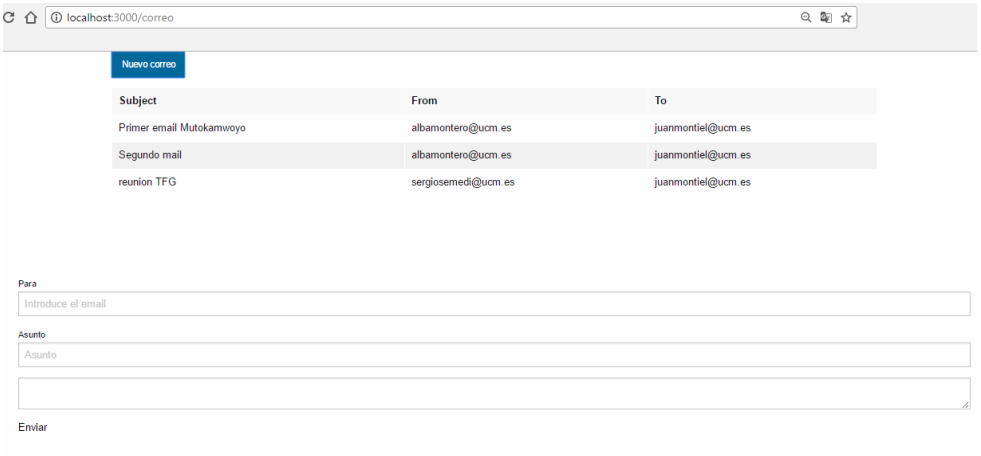


Figura AII.6

Simplemente tenemos que rellenar los campos del formulario y pulsar el botón "Enviar" que se encuentra al final del mismo para enviar el nuevo correo.

MUTOKA MWOYO

Madrid, 8 de mayo de 2017

A quien corresponda,

D. ANDRÉS GONZÁLEZ CERVERA, con DNI 02719911T, coordinador de la asociación Mutoka Mwoyo.

CERTIFICA:

Que «Mutoka Mwoyo Cloud» es un proyecto de voluntariado coordinado por varios estudiantes de la Facultad de Informática de la Universidad Complutense de Madrid que cursan su último año académico de Ingeniería informática (2016-2017).

Que «Mutoka Mwoyo Cloud» tiene como objetivo implementar una red Intranet en el pueblo congoleño de Ngandanjika para poder abastecer de información básica a su población y, en especial, a los médicos del hospital «Virgen de Guadalupe» y a los estudiantes y profesores del colegio «La Robertanna».

Que los estudiantes que trabajan de forma voluntaria en el proyecto «Mutoka Mwoyo Cloud» son los siguientes:

Sergio Alfonso Semedi Barranco (4º de Grado en Ingeniería Informática).

Alba María Montero Monte (4º de Grado en Ingeniería Informática).

Adrián Martínez Jiménez (4º de Grado en Ingeniería Informática).

Juan José Montiel Cano (4º de Grado en Ingeniería Informática).

Que dichos estudiantes, para llevar a cabo el proyecto de «Mutoka Mwoyo Cloud», cuentan con la asistencia y el asesoramiento de los siguientes profesores del Departamento de Arquitectura de Computadores y Automática de la Universidad Complutense de Madrid:

José Luis Vázquez Poletti

José Manuel Velasco Cabo

Y así lo firmo para que conste y surta los efectos oportunos

Fdo.: Andrés González Cervera